AP-484

APPLICATION NOTE

Interfacing a Floppy Disk Drive to an 80C186EX Family Processor

ERIC AUZAS
BRENDEN RUIZ
APPLICATION ENGINEERS

June 1993

Order Number: 272339-001

Information in this document is provided in connection with Intel products. Intel assumes no liability whatsoever, including infringement of any patent or copyright, for sale and use of Intel products except as provided in Intel's Terms and Conditions of Sale for such products.

Intel retains the right to make changes to these specifications at any time, without notice. Microcomputer Products may have minor variations to this specification known as errata.

*Other brands and names are the property of their respective owners.

†Since publication of documents referenced in this document, registration of the Pentium, OverDrive and iCOMP trademarks has been issued to Intel Corporation.

Contact your local Intel sales office or your distributor to obtain the latest specifications before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature, may be obtained from:

Intel Corporation
P.O. Box 7641
Mt. Prospect, IL 60056-7641

or call 1-800-879-4683

Interfacing a Floppy Disk Drive to an 80C186EX Family Processor

CONTENTS

CONTENTS

1.0 INTRODUCTION

Embedded applications often deal with data collection and analysis. Attaching a floppy disk drive to an embedded system allows data collection with the possibility of analyzing the data on a PC Compatible system. This Application Note describes how to interface a 80C186 family processor to a floppy disk drive and read/write in the PC DOS compatible format. The floppy driver does not allow execution of EXE or COM files, it is strictly for data transfer. A simple example of how to use the floppy driver to write random numbers to a DOS file is also provided. The source code and this Application Note are available on the Apps BBS (916-356-3600) as FDCAPP.EXE.

2.0 WHY ADD A FLOPPY DRIVE?

Users often need to easily update their 186 Embedded design with application code, new data, or configuration information. These updates are usually generated by a development computer typically a personal computer (PC). Other possibilities are the need to retrieve data from the embedded controller and analyze it on a desktop PC. The link here is the PC which offers low cost tools for development and easy to use applications for data analysis.

3.0 HARDWARE DESCRIPTION

The hardware interface logic is very simple (Figure 1). The interface uses the 82077AA floppy disk controller, and some additional glue logic. This design is configured for a 3.5″ or 5.25″ drive with a capacity up to 2 MBytes. Order the 82077AA data sheet for complete details on the device. (Intel Literature 1-800-548-4725, Order # 290166-005).

In Figure 1, the EV80C186 CON is the expansion connector from the 80C186XL evaluation board. The address lines are latched and the data lines are buffered. Most control signals come directly from the processor. The EIOS is an I/O chip select generated from the Peripheral Chip Select (PCS) lines of the 80C186XL. The EV80C186 CON is a generic connector easily created in any 80C186 system.

The 82077AA floppy disk controller is configured for AT mode (Pin 27 IDENT), MFM (Pin 48 MFM), active low signals (Pin 35 INVERT), and floppy mode (Pin 39 PLL0). The latched address lines LA1 to LA3 are tied to A0 to A2 causing all even byte address accesses to be transferred on the lower byte of the data bus (D0–D7).

The 82077AA requires either a 24 MHz oscillator or crystal. All the signals between the floppy disk controller and floppy drive are direct connections on AT systems. An exception for PS/2 type systems is DENSEL, which needs to be inverted. Also, pull-up resistors are needed on 5 interface lines (INDEX, DSKCG, RDATA, WP, TRK00), the required resistance is 1 KΩ for both 3.5″ and 5.25″ drives.

3.1 82078 Floppy Disk Controller

A new addition to the Floppy Disk Controllers is the 82078, which is functionally compatible with the 82077AA/SL (software transparent) but includes the following additional features:

— 3.3V or 5V operation

— Small QFP package 44- or 64-pin

— New 2 Mbps data rate for tape drives

— An enhanced command set

The 82078 is not pin for pin compatible with the 82077AA/SL but maintains the essential pins for providing a PC compatible floppy disk controller. A note in Appendix B describes how to convert 82077AA/SL designs to 82078 designs.

3.2 Non-DMA Data Transfers

If DMA transfers are not possible then either interrupt or polling can be used for the transfer. For interrupt driven systems there is an INT pin and for polling systems the RQM bits in the Main Status Register indicate data available. The 16 byte FIFO would allow the use of the OUTS (out string) or INS (in string) instructions. These non-DMA transfers still require TC and DACK logic to terminate the transfer. It is possible not to use TC and DACK but a terminate error will occur when reading back the results of the transfer. This error can just be ignored if no terminate logic is being implemented.

3.3 DMA Data Transfers

This design uses DMA to transfer the data to and from the disk, allowing other routines to run between transfers. The time between transfers is 10 μs to 14.5 μs for a 1.44 MB 3.5″ drive. The 80C186 DMA controller uses two bus cycles for a transfer, the first cycle is the fetch cycle and the second is the deposit cycle. Since the 80C186 DMA does not support a pin for Terminal Count or DMA Acknowledge, address decode logic was used to generate these (Figure 1). The TC and DACK I/O locations and the 82077AA register locations are shown in Table 1.

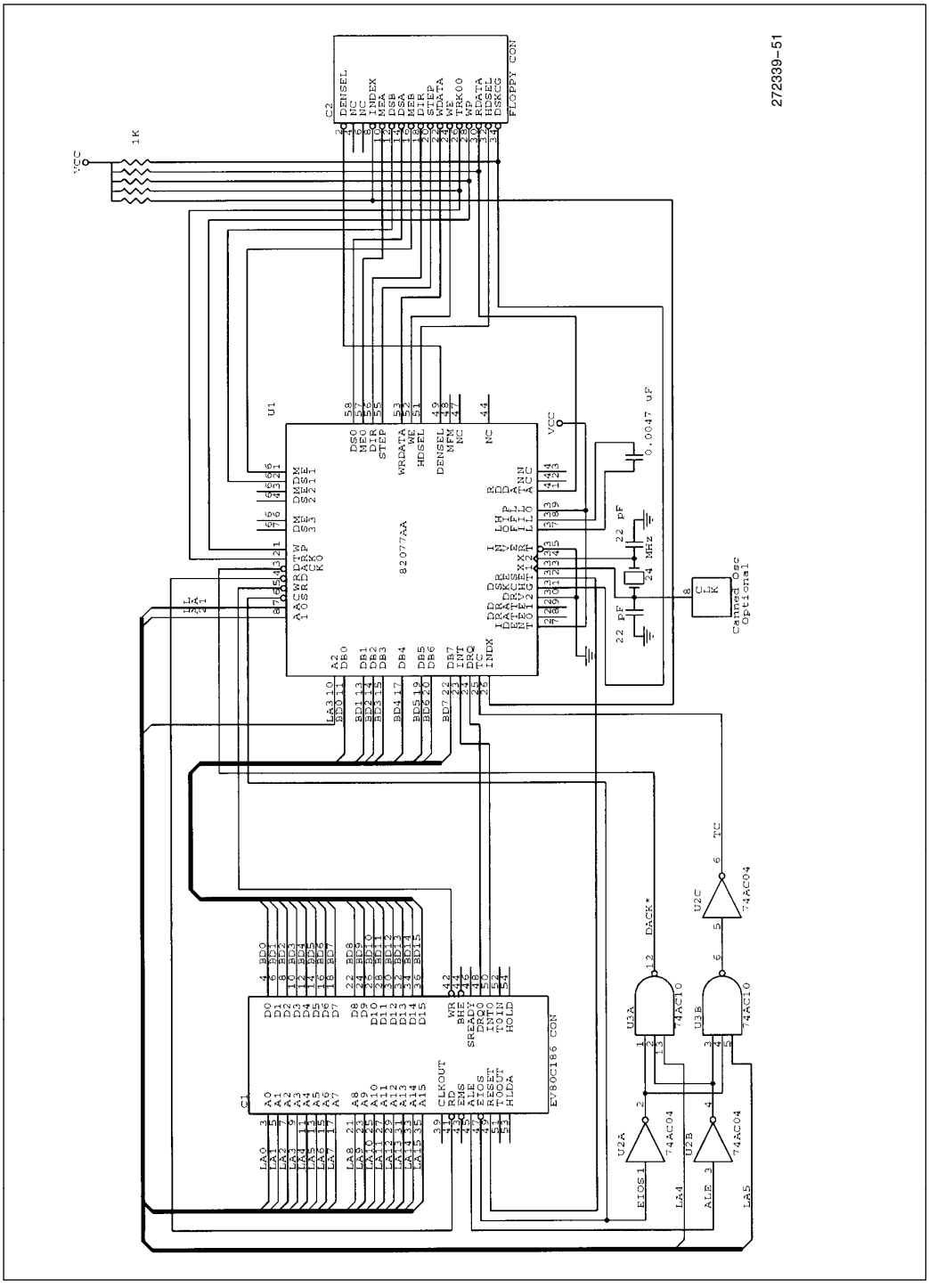

Figure 1. XL/EA Floppy Drive Expansion Interface

Table 1. 82077AA Register Locations

I/O Address	R/W	Register
XX00H	R	Status Register A
XX02H	R	Status Register B
XX04H	R/W	Digital Output Register
XX06H	R/W	Tape Drive Register
XX08H	R	Main Status Register
XX08H	W	Data Rate Select Register
XX0AH	R/W	Data (FIFO)
XX0CH		Reserved
XX0EH	R	Digital Input Register
XX0EH	W	Configuration Control Register
XX1AH	W	DMA DACK Port
XX3AH	W	DMA TC Port

DMA Acknowledge (DACK#) is active low and is generated when address line A4 is high, I/O PCS is low, and data is on the bus (ALE is low and Data Enable #DEN is low). Terminal Count (TC) must be sent before the controller timeout of 6 μs. TC is active high and is generated when address line A5 is high, I/O PCS is low, and data is present on the bus (ALE is low and Data Enable is low). Keep in mind, to signal the 82077AA when a DMA transfer is terminated, both DACK and TC need to be active for a minimum of 50 ns, therefore the address is XX3AH for the TC port (Table 1), setting both lines active.

TC needs to be asserted when the last byte of the DMA transfer is sent. This can be done using an interrupt routine, a timer, or polling the STRT bit of the DxCON register. The first method, using an interrupt routine, requires that an interrupt be generated at the end of the transfer. The transfer count would need to be programmed as N-1, where N is the total amount of transfer bytes. Once the interrupt is generated, there is an interrupt latency of 42 clock cycles or 2.1 μs at 20 MHz before the first instruction of the Interrupt Service Routine (ISR) executes. The 82077AA delays about 10 μs to 14.5 μs between byte transfers. This should be plenty of time to send out the last byte with the terminal count. The second method, involves using a timer to count DRQ or DACK# pulses. The amount of data to be transferred is written into the Timer Count register and once the count is reached a Terminal Count pulse is sent out to the timer output pin. The third method, is to poll the STRT bit of the DMA control register until all the data has been transferred except the last byte and then output a Terminal Count pulse through an I/O port.

The Floppy Driver uses the polling method, but any of the above methods could be used. When sending out the last byte, the timing must be correct so TC is sent after DRQ is active. The program may have to delay before sending out TC, waiting for DRQ from the 82077AA.

The timing analysis includes signals from the 80C186 and the 82077AA. Figure 2 demonstrates compliance for T_{25} (82077AA Data Sheet) between the RD#/WR# lines and DACK#.

T_1 = 50 ns (nanoseconds) (Clock Period)

T_{CHLL} = 20 ns (ALE Inactive Delay)

T_{CVCTV} = 3 ns min, 25 ns max (Control Active Delay)

$T_{DACKPROP}$ = U2A + U3A = 7.5 ns + 6.5 ns = 14 ns (Gate Propagation Delay)

T_{25} = 5 ns min (DACK setup to RD, WR)

$T_{DACK} = T_{CHLL} + T_{DACKTPROP}$ = 20 ns + 14 ns = 34 ns

Looking at the maximum values DACK would go active 9 ns after T_1 and the RD/WR line would go active 25 ns after T_1. This allows for a margin of 16 ns when the specification calls for a minimum of 5 ns.

After the 82077AA asserts DRQ for a DMA request, the 80C186 must respond with a READ or WRITE to the controller within 6 μs (Figure 3, T_{27}), otherwise a timeout error will result. This only becomes critical when writing the last byte with terminal count because it is not accomplished by the DMA controller.

T_{27} = 6 μs max (DRQ to RD, WR Active)

Two types of DMA transfers are used: Source Synchronized DMA Transfers and Destination Synchronized DMA Transfers. Source Synchronized DMA Transfers are used when the Source controls the transfer. In this case, the controller is the source and performs a read (Fetch Cycle-4 clocks) from the disk drive and writes (Deposit Cycle-4 clocks) to memory. This type of trans-

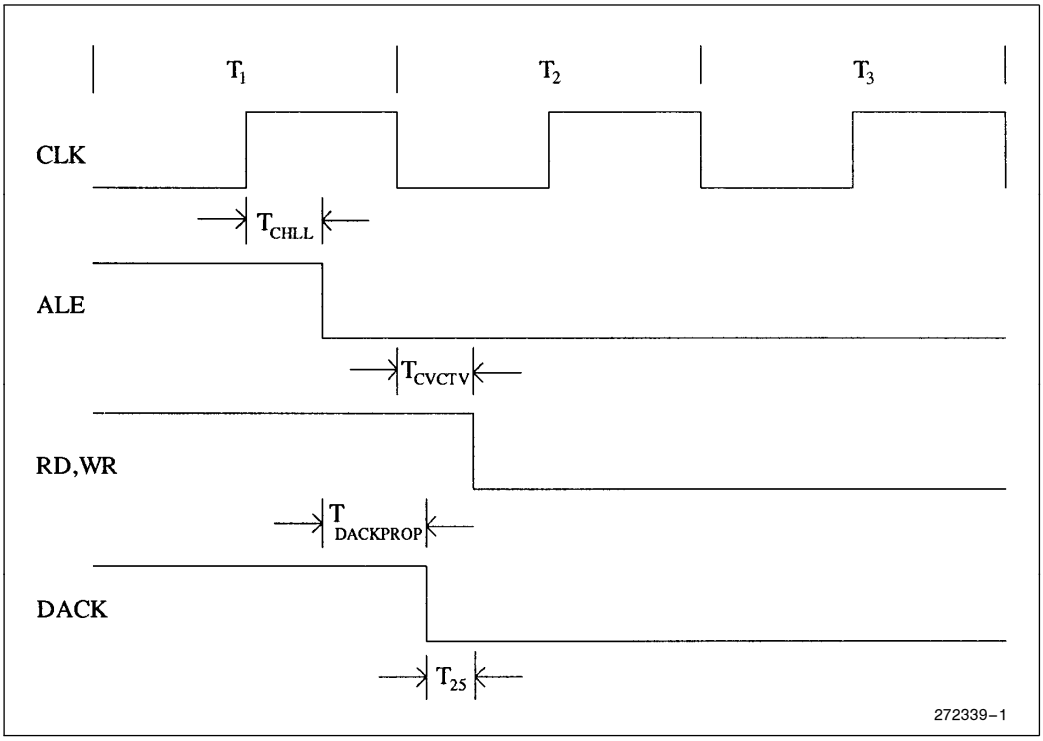

Figure 2

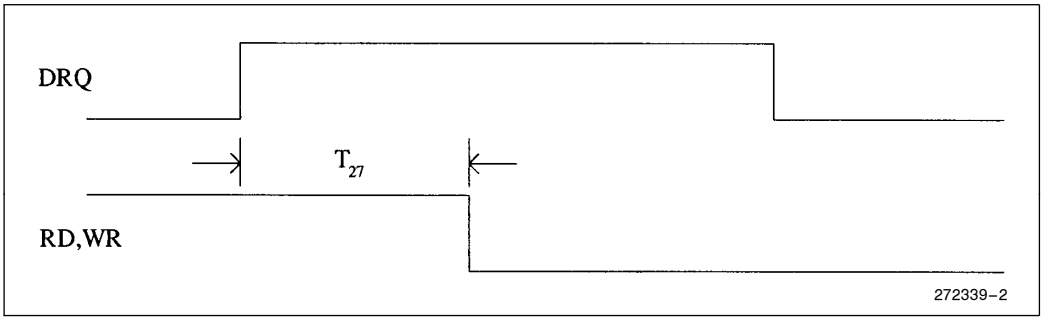

Figure 3

fer requires the DRQ signal to be deasserted at least 4 clocks before the end of the DMA transfer (at the T_1 state of the deposit phase) to prevent another DMA cycle. The access to the disk drive causes the DACK line to go active and the 82077AA deasserts its DRQ line. This transfer method provides the source device almost three clock cycles from when it is accessed (acknowledged) to deassert its request line before the next transfer is to take place. The DMA acknowledge occurs 9 ns after T_2 and the maximum time response for the 82077AA to deassert DRQ is 75 ns, therefore DRQ is guaranteed to be deasserted in T_3, well before T_1 of the Deposit Cycle (Figure 4).

T_{23} = 75 ns max (DACK to DRQ Inactive)

$T_{DACK} = T_{CHLL} + T_{DACKPROP} - \frac{1}{2}T_1$
$= 20\text{ ns} + 14\text{ ns} - 25\text{ ns} = 9\text{ ns}$

$T_{DRQ} = T_{DACK} + T_{23}$
$= 9 + 75 = 84$ max from T_2

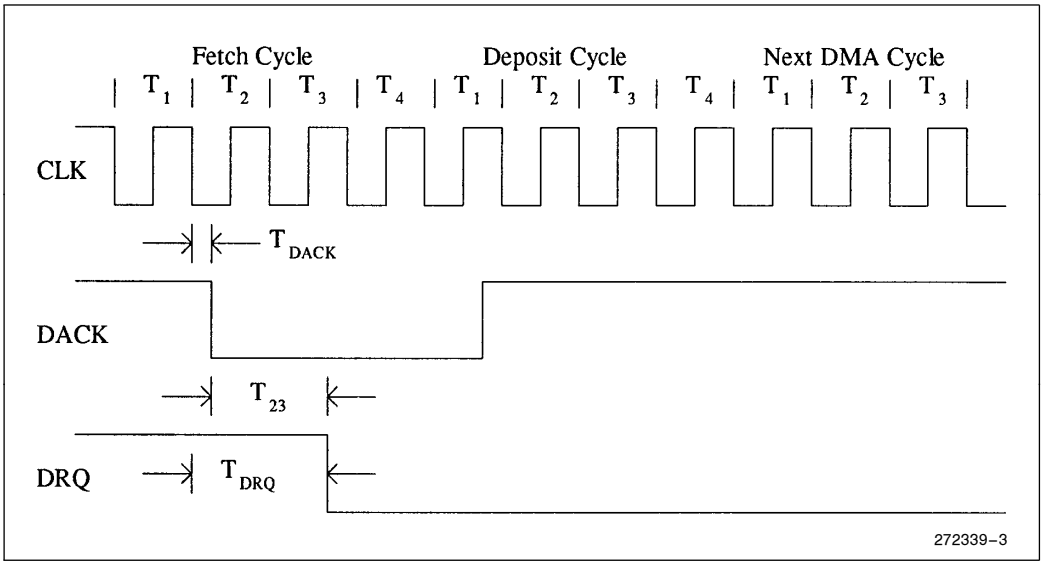

Figure 4. Source Synchronized Transfer

When using Destination Synchronized Transfers the floppy controller becomes the destination. The DMA controller inserts two idle cycles at the end of the deposit cycle when using Destination Synchronized Transfer. In this type of transfer, the fetch cycle is from memory and the deposit cycle is to the 82077AA controller. This causes a problem because the DMA requesting device (82077AA) will not receive the acknowledge until about 5 cycles before the end of the transfer and DRQ will go low 75 ns later. The DMA controller requires DRQ to be deasserted at least 4 clock cycles prior to the next DMA cycle to determine if another transfer takes place. This means that DRQ must be deasserted before the end of T_2 and the DRQ setup time (Figure 5) if no wait states are used.

Figure 5 Destination Synchronized DMA Transfer

T_{SETUP} = 10 ns (20 MHz 80C186XL)

T_2 = 50 ns (Period at 20 MHz operation)

$T_{MAX} = T_2 - T_{SETUP} + (WAIT * T_2)$

Table 2

Number of Wait States WAIT	T_{MAX} (ns) Max Time for DRQ Inactive from Start of T_2
0	40
1	90
2	140
3	190

In using the 82077AA, the maximum time until DRQ (T_{DRQ}) is released is 84 ns before the start of T_2. This puts DRQ going low during T_3 which allows three T states before the next bus cycle (Figure 6). Checking T_{MAX} from Table 2, one waitstate will be needed for correct operation.

T_{TCPROP} = U2B + U3B + U2C
= 7.5 ns + 6.5 ns + 7.5 ns
= 21.5 ns (Gate Propagation Delay)

T_{23} = 75 ns max (DACK to DRQ Inactive)

T_{28} = 50 ns min (Terminal Count Width)

$T_{DACK} = T_{CHLL} + T_{DACKPROP} - \frac{1}{2}T_1$
= 20 ns + 14 ns − 25 ns = 9 ns

$T_{DRQ} = T_{DACK} + T_{23} = 9 + 75 = 84$ max

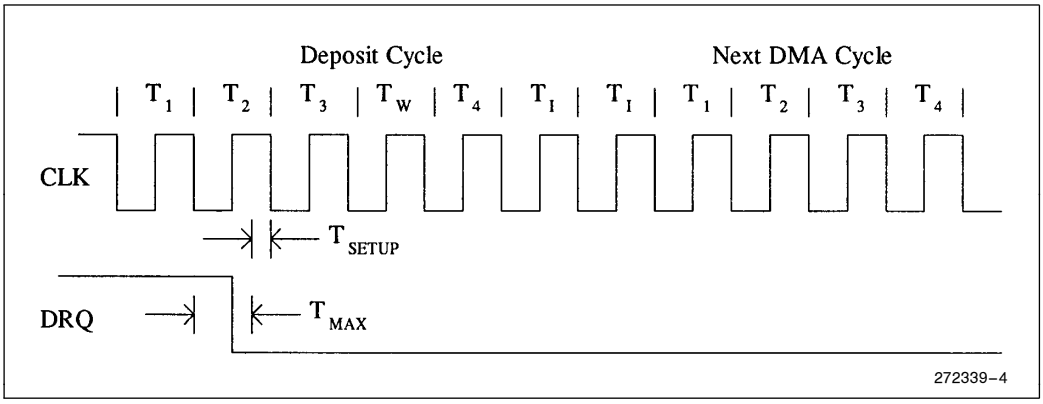

Figure 5. Destination Synchronized DMA Transfer

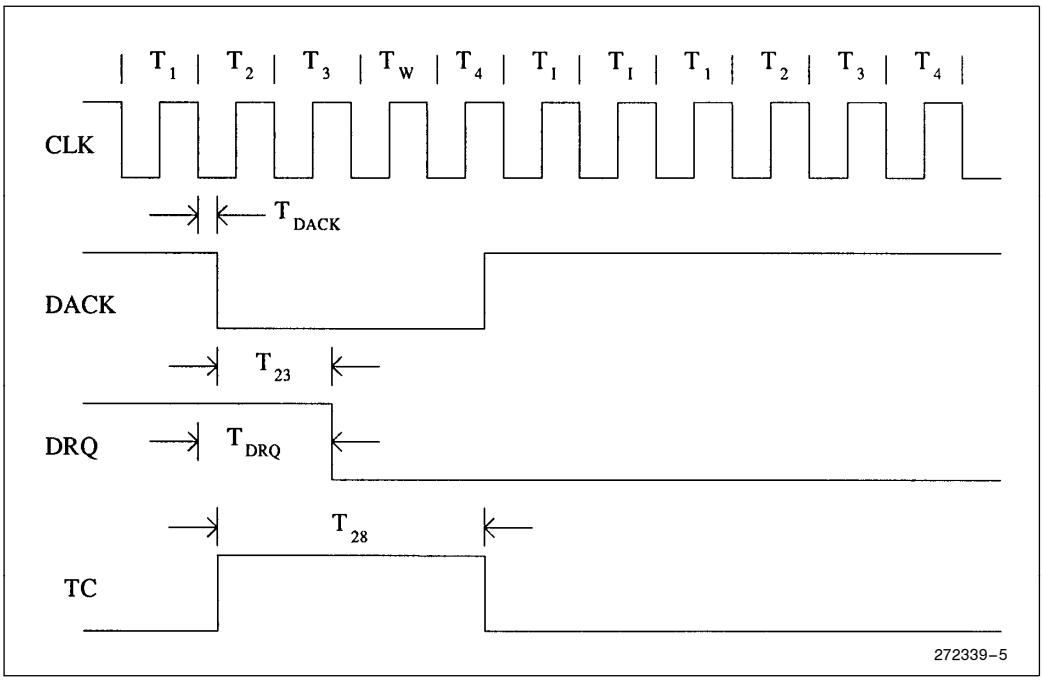

Figure 6

4.0 SOFTWARE

The Floppy Drive Interface software is more complicated and consists of three parts:

1. APPLICATION
2. FLOPPY DISK OPERATING SYSTEM
3. FLOPPY DISK DRIVER

4.1 Floppy Disk Driver

The 82077AA has a built-in command set providing a basic interface to the controller. The commands are outlined in Table 3 and are used to generate the Floppy Disk Driver command set, identical to the PC BIOS Floppy Disk command set (Appendix A).

Table 3. 82077AA Command Set

Command	Description
Read Data	Read a sector
Read Deleted Data	
Write Data	
Write Deleted Data	
Read Track	
Verify	
Format Track	Formats an entire cylinder
Recalibrate	Head retracted to track 0
Sense Interrupt Status	Returns Status Information after each seek operation
Specify	
Sense Drive Status	Status information about FDD
Seek	Head is positioned over proper cylinder or diskette
Configure	Setup Information
Relative Seek	
Dumpreg	Debug info
Read ID	Reads cylinder ID information
Perpendicular Mode	
Lock	
Invalid	Any invalid command codes fall here

Before any of these commands can be issued, the 82077AA must be initialized according to the Initialization Sequence found in the 82077AA data sheet. The sequence is sent to the controller every time a Floppy disk BIOS driver command is issued. The initialization sets up parameters for the size and type of floppy drive being used. After initializing the controller, any command can be sent to the 82077AA. Flowcharts in the data sheet describe the routines to do a read/write operation, initialize option, formatting, recalibration, seeking, and byte fetching. The basic commands allow for error checking via 4 status registers.

The floppy disk driver programs the 82077AA to provide basic disk BIOS operations:

RESET DISK SYSTEM
GET DISK SYSTEM STATUS
READ SECTOR
WRITE SECTOR
VERIFY SECTOR
FORMAT TRACK
READ DRIVE PARAMETERS
READ DISK TYPE
CHECK FOR DISK CHANGE
SET DISK TYPE
SET MEDIA TYPE

The driver used in this system is originally from Annabooks but was modified to run on a non-IBM PC system. The Floppy driver is interrupt driven like the Floppy BIOS on a PC. The routines provide floppy controller setup, command decode, floppy controller reset, seek, motor control, DMA setup, recalibrate head, command results, media change, and disk change. These routines provide the basic disk operations outlined above. Like a PC environment the registers are loaded with command information and a software interrupt is issued to call the driver. The register definitions for each of the above commands are defined in Appendix A.

The driver was written to use DMA but it is not necessary. If DMA is not needed then two methods can be used to transfer the data. One would be to poll the RQM bits of the 82077AA, the other is to use an interrupt service routine. The 82077AA provides a FIFO of up to 16 bytes, which is useful if using the OUTS or INS instructions to implement the transfer, otherwise byte transfers are possible. The driver is called using a software interrupt, after loading the appropriate function number and data into the registers. Appendix A lists the various functions available.

4.2 Floppy Disk Operating System

With these drivers, it is possible to read and write MS-DOS type diskettes. The operating system needs to determine what type of floppy is to be read or written. This can be done with the BIOS service Get Disk Type, Table 4 is a listing of the possible media types.

To achieve this, one must understand the File Allocation Table (FAT), Boot Sector, and Directory Table. At offset zero is the JMP instruction, which can either be a 16-bit or 8-bit displacement If neither of these two jumps are found then the disk has not been formatted. Once a disk is found that is formatted, the operating system needs to determine location of the FATs and the Directory Table.

Table 4. Media Types

Capacity	Size	Heads	Tracks	Sectors	Bytes/Cluster
160K	5.25″	1	40	8	512
180K	5.25″	1	40	9	512
320K	5.25″	2	40	8	1024
360K	5.25″	2	40	9	1024
1.2M	5.25″	2	80	15	512
720K	3.5″	2	80	9	1024
1.4M	3.5″	2	80	18	512

Table 5. Boot Sector

Offset	Mnemonic	Description	DOS
00H		E9 XX XX or EB XX 90JMP Code (3 bytes)	
03H		OEM Name and Version (8 bytes)	
0BH	BYTES__SEC	Bytes per Sector (2 bytes)	2.0
0DH	SEC__CLUS	Sectors per Cluster (1 byte)	2.0
0EH	RES__SEC	Reserved Sectors, Starting at 0 (2 bytes)	2.0
10H	NUM__FATS	Number of FATs (1 byte)	2.0
11H	NUM__ROOT__DIR__ENT	Number of Root Directory Entries (2 bytes)	2.0
13H		Total Sectors in Logical Volume (2 bytes)	2.0
15H	MEDIA__DESC	Media Descriptor (1 byte)	2.0
16H	NUM__SEC__FAT	Number of Sectors per FAT (2 bytes)	2.0
18H	SEC__TRK	Sectors per Track (2 bytes)	3.0
1AH	NUM__HEADS	Number of Heads (2 bytes)	3.0
1CH	NUM__HID__SEC	Number of Hidden Sectors (4 bytes)	3.0/4.0
20H–3D		Additional Features Byte	4.0
3EH		Bootstrap	

These can be calculated from the boot sector (Table 5). Since the boot sector is located at sector one, the first FAT must begin at sector 2. The length of the FAT is determined by the word at offset 16H Number of Sectors per FAT. If there is more than one FAT, offset 10H Number of FATs, then its beginning location must be determined. Next the directory table offset and length is calculated. Since it follows the Boot Sector and FATs just add one to their sum to calculate the starting sector. The directory length depends on the Number of Root Directory Entries multiplied by 32, the length in bytes for each entry, and divided by the number of Bytes per Sector, most commonly 512. Table 6 illustrates the sector layout for a typical floppy disk:

Table 6. Sector Layout for DOS

Starting Sector	Length	Description
0	1 (Note 1)	Boot Sector
1	NUM_SEC_FAT	FAT 1
NUM_SEC_FAT + 1	NUM_SEC_FAT	FAT 2(2)
2*NUM_SEC_FAT + 1	(NUM_ROOT_DIR_ENT * 32)/BYTES_SEC	Directory Table
After Directory Table	Remaining Sectors of Disk	File Data

NOTES:
1. The length of the boot sector is part of a reserved area that can actually be extended by the RES_SEC location. Normally a 1 is located here for the boot sector.
2. The location NUM_FATS indicates if there is a second FAT.

Once the locations have been calculated it is important to determine the type of media being read. This can be accomplished by reading the media descriptor byte and comparing it with Table 7.

Table 7. Disk Descriptors

Descriptor	Medium	DOS Version
0F0H	3.5″ Floppy, 2 Sided, 18-Sector	3.3
0F8H	Fixed Disk	2.0
0F9H	5.25″ Floppy, 2 Sided, 15-Sector 3.5″ Floppy, 2 Sided, 9-Sector	3.0 3.2
0FCH	5.25″ Floppy, 2 Sided, 9-Sector	2.0
0FDH	5.25″ Floppy, 2 Sided, 9-Sector	2.0
0FEH	5.25″ Floppy, 1 Sided, 8-Sector	1.0
0FFH	5.25″ Floppy, 2 Sided, 8-Sector	1.1

The FAT is located after the boot sector and due to its importance there may be a duplicate copy following the first one. At the first location of the FAT is a duplicate copy of the media descriptor byte as found in the boot sector. Floppy disks are usually under 16 meg, and therefore require only a 12-bit FAT. This allows up to 4096 clusters, (Table 8) with up to 8 sectors per cluster, and 512 bytes per sector. Using a 12-bit FAT requires some decoding of the bytes since every three bytes represents two cluster locations. The cluster size for floppies under 2 meg is 1 sector per cluster. Therefore the cluster location is a direct correlation to the sector location.

Table 8. File Allocation Table (FAT)

Offset	Description
0	Media Descriptor Byte
1	FF FF
3	Beginning of Fat Entries

To calculate the cluster location for the three bytes, use the following equations:

Cluster Entry 1 = (Byte2 AND 0FH) * 100H + Byte 1
Cluster Entry 2 = (Byte3 * 10H) + (Byte2 AND 0F0H) / 10H

Table 9 describes the possible 12-bit byte assignments for determining if a cluster is free, reserved, bad or the end of the cluster chain.

Table 9. Cluster Definition for the FAT

Description	12-Bit Code
Free of assignment	0
Part of a file (Pointers to next clusters)	2–FEF
Reserved	FF0–FF6
Bad cluster	FF7
End of cluster chain	FF8–FFF

The directory entries follow the format of Table 10. The starting cluster is the location in the FAT table where the first cluster of the file can be found. The cluster locations are actually an offset into the file data area which begins after the directory area. The file size is described in bytes. The notes describe the format of the directory entries.

Table 10. Directory Format

00H	Filename (8 bytes)	(Note 1)
08H	Extension (3 bytes)	
0BH	File attribute (1 byte)	(Note 2)
0CH	Reserved (10 bytes)	
16H	Time created or last updated (2 bytes)	(Note 3)
18H	Data created or last updated (2 bytes)	(Note 4)
1AH	Starting cluster (2 bytes)	
1CH	File Size (4 bytes)	

Note 1 for Table 10

Value	Meaning
00H	Directory entry has never been used; end of occupied portion of directory
05H	First character of filename is actually E5H
2EH	Entry is an alias for the current or parent directory. If the next byte is also 2EH, the cluster field contains the cluster number of the parent directory (zero if the parent directory is the root directory).
E5H	File has been erased

Note 2 for Table 10

Bit	Meaning
0	Read-only; attempts to open file for write or to delete file will fail.
1	Hidden file; excluded from normal searches.
2	System file; excluded from normal searches.
3	Volume label; can exist only in root directory.
4	Directory; excluded from normal searches.
5	Archive bit; set whenever file is modified.
6	Reserved
7	Reserved

Note 3 for Table 10

Bits	Contents
00H–04H	Day of month (1–31)
05H–08H	Month (1–12)
09H–0FH	Year (relative to 1980)

5.0 CODE CONSIDERATIONS

The Floppy Disk Driver code from Annabooks was modified for the 80C186 family. There is no fee, or royalty for this software but Annabooks does require users to register with them before using it. The example program provided, writes random numbers to a DOS compatible file. It is only a very simple example and does not provide complete error checking.

Annabooks Inc.
15010 Ave. of Science Ste. 101
San Diego, CA 92128
Tel: (619) 271-9526
Fax: (619) 673-1432

APPENDIX A
FLOPPY DISK BIOS FUNCTION CALLS

RESET DISK SYSTEM

Call With
AH = 00H
DL = Drive

Returns
If function successful
Carry flag = clear
AH = 00H
If function unsuccessful
Carry flag = set
AH = status flags

GET SYSTEM STATUS

Call With
AH = 01H
DL = Drive

Returns
AH = 00H
AL = Status of previous disk operation
- 00H = no error
- 01H = invalid command
- 02H = address mark not found
- 03H = disk write protected
- 04H = sector not found
- 06H = floppy disk removed
- 08H = DMA overrun
- 09H = DMA crossed 64 KB boundary
- 0CH = media type not found
- 10H = uncorrectable CRC or ECC data error
- 20H = controller failed
- 40H = seek failed
- 80H = disk timeout (failed to respond)

READ SECTOR

Call With
AH = 02H
AL = number of sectors
CH = cylinder
CL = sector
DH = head
DL = drive
- 00H–7FH floppy disk

ES:BX = segment:offset of buffer

Returns:
If function successful
Carry flag = clear
AH = 00H
AL = number of sectors transferred
If function unsuccessful
Carry flag = set
AH = status

WRITE SECTOR

Call with
AH = 03H
AL = number of sectors
CH = cylinder
CL = sector
DH = head
DL = drive
00H–7FH floppy disk
ES:BX = segment:offset of buffer

Returns
If function successful
Carry flag = clear
AH = 00H
AL = number of sectors transferred
If function unsuccessful
Carry flag = set
AH = status

VERIFY SECTOR

Call with
AH = 04H
AL = number of sectors
CH = cylinder
CL = sector
DH = head
DL = drive
00H–7FH floppy disk
ES:BX = segment:offset of buffer

Returns
If function successful
Carry flag = clear
AH = 00H
AL = number of sectors verified
If function unsuccessful
Carry flag = set
AH = status

FORMAT TRACK

Call with
AH = 05H
CH = cylinder
CL = sector
DH = head
DL = drive
00H–7FH floppy disk
ES:BX = segment:offset of address field

Returns
If function successful
Carry flag = clear
AH = 00H
AL = number of sectors verified
If function unsuccessful
Carry flag = set
AH = status
Address field:

Byte	Contents
0	cylinder
1	head
2	sector
3	sector size code 00H if 128 bytes per sector 01H if 256 bytes per sector 02H if 512 bytes per sector (PC standard) 03H if 1024 bytes per sector

GET DRIVE PARAMETERS

Call with: AH = 08H
DL = drive
00H–7FH floppy disk

Returns: If function successful
Carry flag = clear
BL = drive type
01H = 360 KB, 40 track, 5.25″
02H = 1.2 MB, 80 track, 5.25″
03H = 720 KB, 80 track, 3.5″
04H = 1.44 MB, 80 track, 3.5″
CH = low 8 bits of maximum cylinder number
CL = bits 6–7 high order 2 bits of maximum cylinder number
bits 0–5 maximum sector number
DH = maximum head number
DL = number of drives
ES:DI = segment:offset of disk drive parameter table
If function unsuccessful
Carry flag = set
AH = status

GET DISK TYPE

Call with: AH = 15H
DL = drive
00H–7FH floppy disk

Returns: If function successful
Carry flag = clear
AH = drive type code
00H = no drive present
01H = floppy disk drive without change-line support
02H = floppy disk drive with change-line support
If function unsuccessful
Carry flag = set
AH = status

GET DISK CHANGE STATUS

Call with AH = 16H
DL = drive

Returns If change line inactive (disk has not been changed)
Carry flag = clear
AH = 00H
If change line active (disk may have been changed)
Carry flag = set
AH = 06H

SET DISK TYPE

Call with AH = I7H
AL = floppy type code
00H = not used
01H = 320/360 KB floppy disk in 360 KB drive
02H = 320/360 KB floppy disk in 1.2 MB drive
03H = 1.2 MB floppy disk in 1.2 MB drive
04H = 720 KB floppy disk in 720 KB drive
DL = drive
00H–7F floppy drive

Returns If function successful
Carry flag = clear
AH = 00H
If function unsuccessful
Carry flag = set
AH = status

SET MEDIA TYPE FOR FORMAT

Call with AH = 18H
CH = number of cylinders
CL = sectors per track
DL = drive
00H–7F floppy drive

Returns If function successful
Carry flag = clear
AH = 00H
ES:DI = segment:offset of disk parameter table for media type
If function unsuccessful
Carry flag = set
AH = status

APPENDIX B
CONVERTING 82077SL/AA DESIGNS TO 82078 DESIGNS

Purpose:

This note describes the design changes needed to replace the 82077SL/AA floppy controller designs with the new 44- and 64-pin 82078 floppy controller. Also included are full schematics for implementing the 82078 on the ISA bus.

Introduction:

The 82078 is the next generation of floppy controllers from Intel. It is functionally compatible with the 82077AA/SL (software transparent) and includes additional features to support today's new smaller low voltage platforms. Some of the features are:

— 3.3V operation

— Small QFP package (44-pin and 64-pin)

— New 2 Mbps data rate for tape drives

— An enhanced command set

The 82078 is available in 44-and 64-pin packages. Several pin changes accommodate the reduced pin count (from the 68-pin 82077SL) and the added features. The 44-pin part provides a low cost solution for the 5.0V ISA/EISA market. The 64-pin part features 3.3V operation, 2 Mbps data rate, and accommodates the ISA/EISA, MCA markets. This memo describes how to design a 82078 in current 82077SL/AA and compatible designs. It also briefly describes some of the new 82078 features.

Replacing the 82077SL with a 82078 at 5.0V:

The 82078 easily replaces the 5.0V 82077SL with minimum design changes. Figure 7 shows a PC/AT design using the 82077SL; Figure 8 shows the same design using the 82078.

The connections to the AT bus are the same in both designs. MFM and IDENT have been changed to IDENT1 and IDENT0 (see *Pin Changes on the 64-Pin Part*). PLL0 was removed. Like the 82077SL, configure the tape drive mode via the Tape Drive Register (TDR).

The 82078 connection to the Disk Interface is similar to the 82077SL. DRVDEN0 and DRVDEN1 on the 82078 take the place of DENSEL and DRATE0 on the 82077SL. The Drive Specification Command configures each drive via these pins. The Motor Enable pins (ME0–3) and the Drive Select pins (DS0–3) are renamed FDME(0–3) and FDS(0–3) respectively on the 82078. 10K pull-up resistors can be used on the disk interface.

Replacing the 82077SL with a 82078 at 3.3V:

The design for 3.3V is the same as for 5.0V with two exceptions: The SEL3V# pin must be held low to select 3.3V operation, and the VCCB pin can be either 3.3V or 5.0V (VCCB can only be 5.0V when SEL3VT is high).

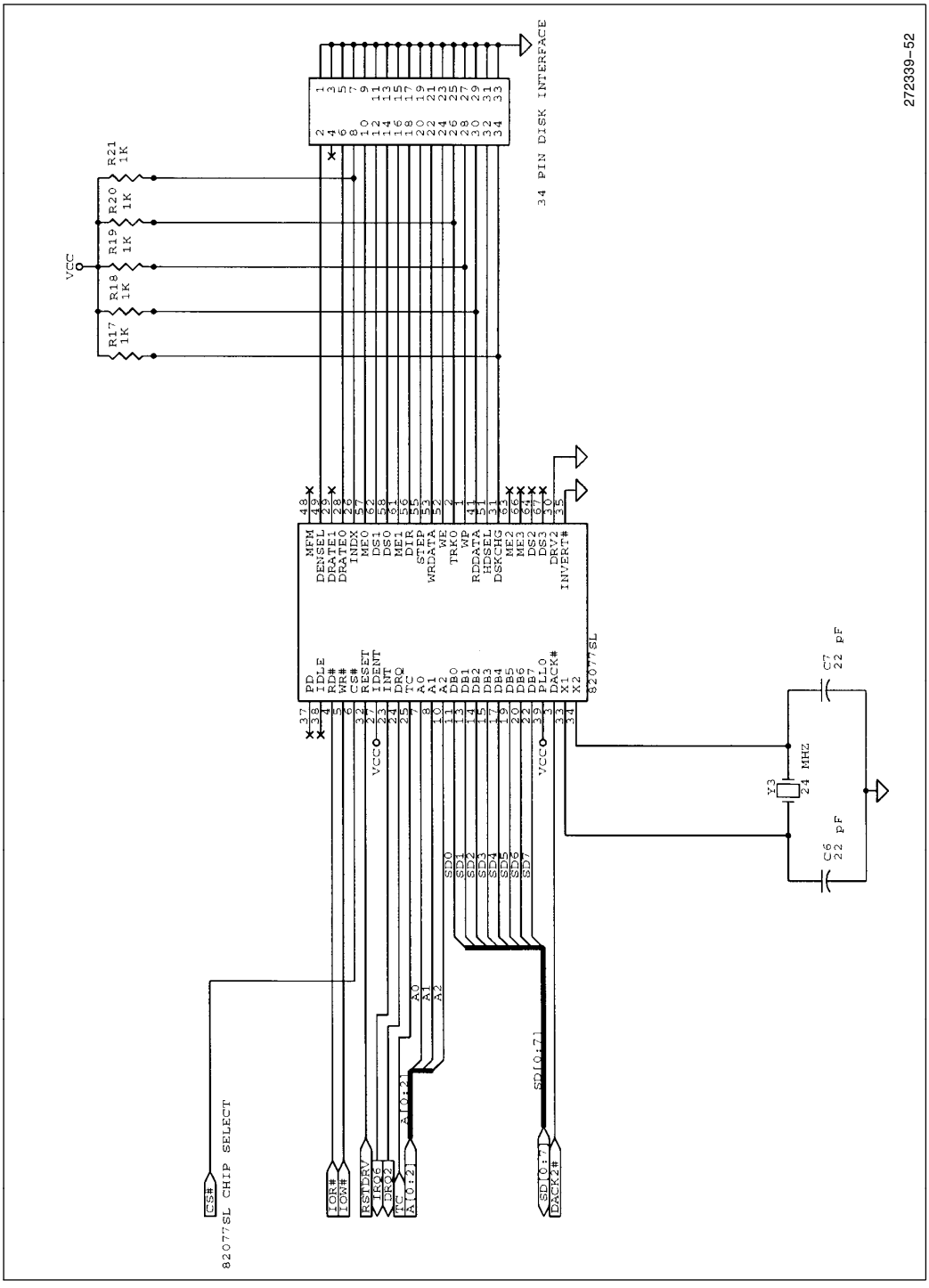

Figure 7. 82077SL PC/AT Floppy Disk Controller

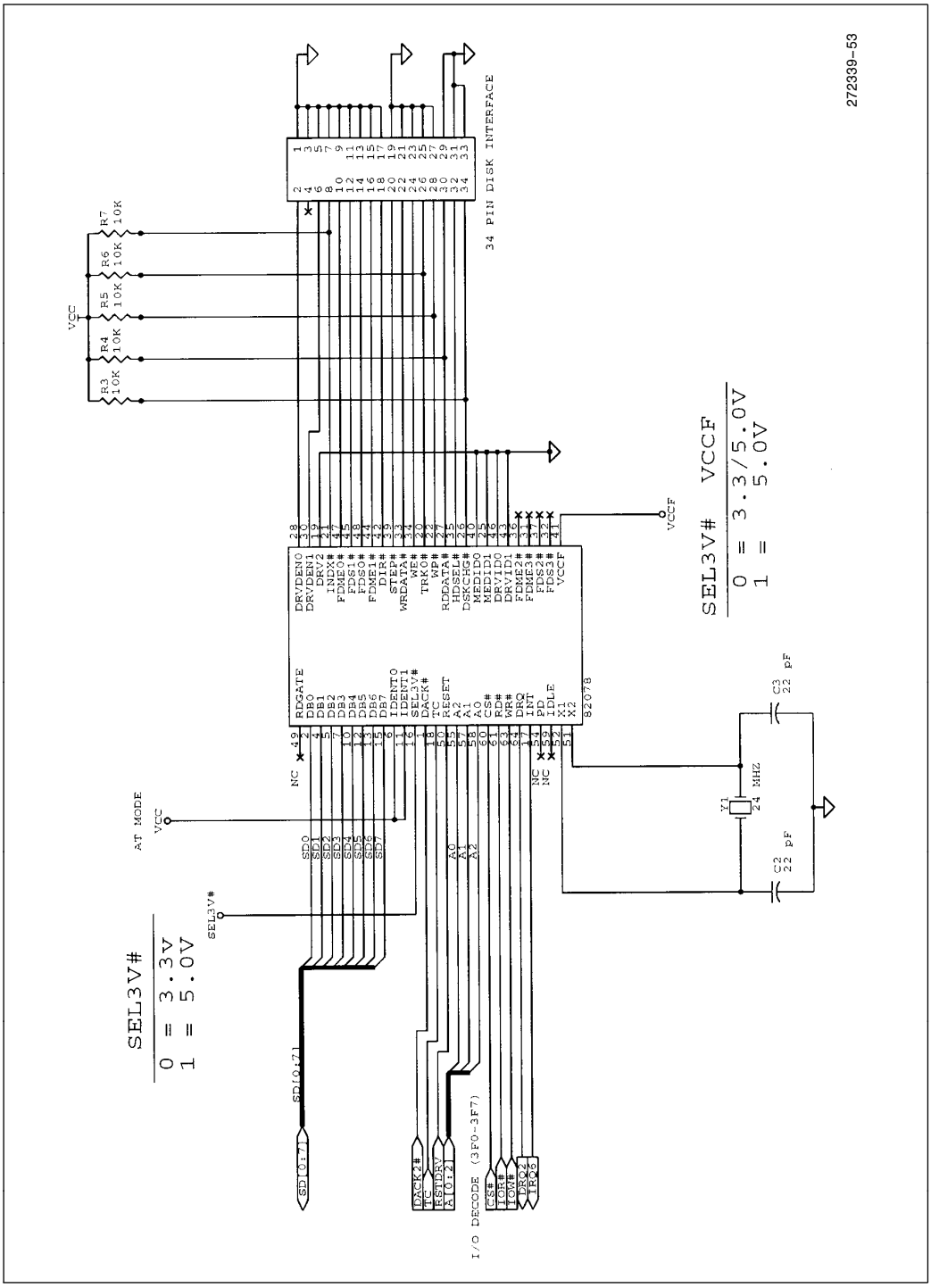

Figure 8. 82078 (64-Pin) Conversion to 82077SL Design

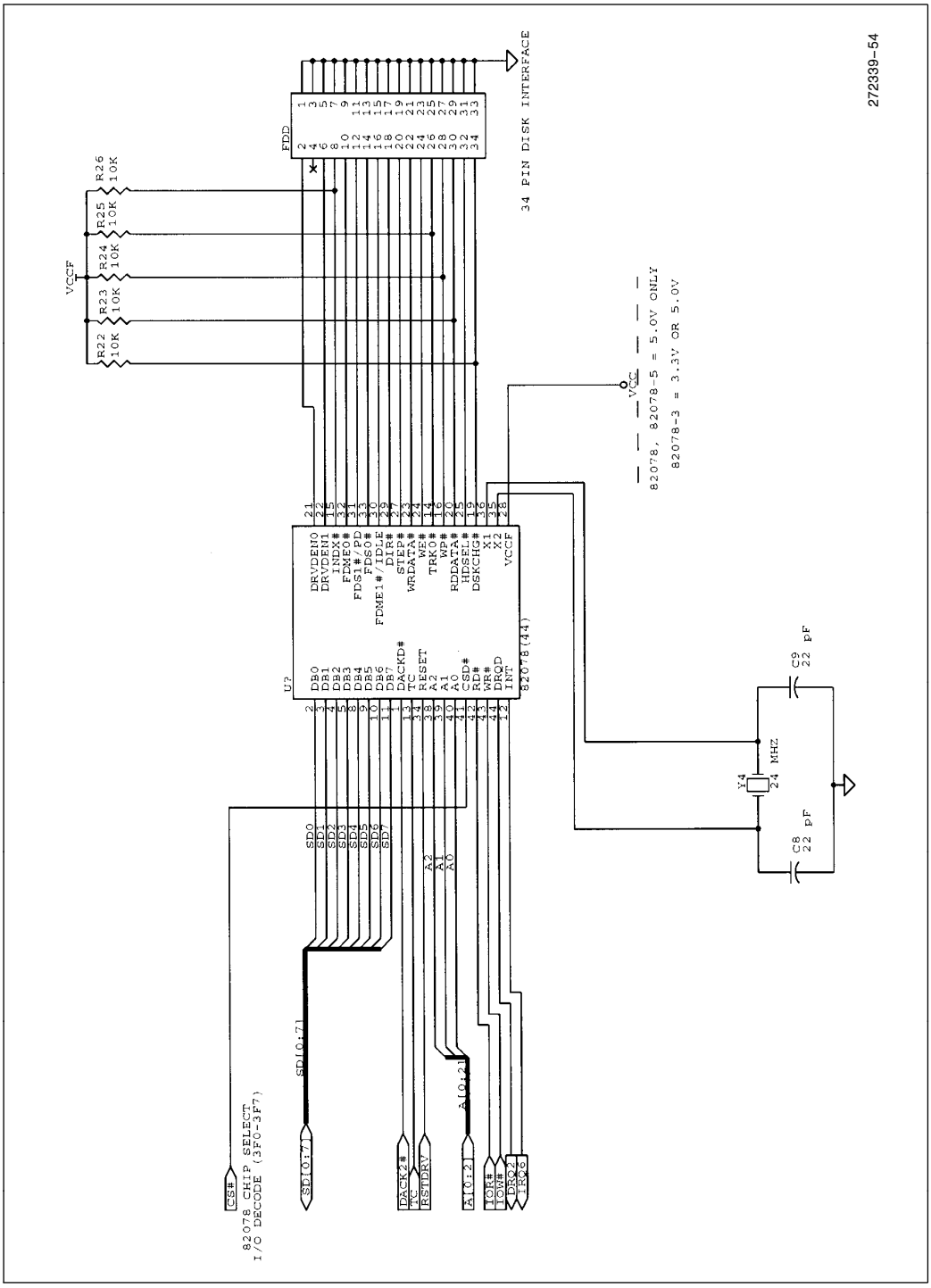

Figure 9. 82078 (44-Pin) PC/AT Design

Command Enhancements:

The 82078 supports AT, EISA, and MCA market requirements, it also supports an enhanced feature set. The following table lists the added features of the 64- and 44-pin parts.

Feature	82078 (44-Pin)	82078 (64-Pin)
# of Drives Supported	2	4
Architecture Supported	AT/EISA Only	AT/EISA/MCA
PD and IDLE Pins	Multiplexed PD & IDLE	Yes
3.3V Support	No	Yes
Enhanced TDR Support	Yes	Yes
Drive Specification Command	Yes	Yes
Media ID Pins/Support	No	Yes
2 Mbps Tape Standard	No	Yes
Enhanced PD Command	Yes	Yes
Part ID Command	Yes	Yes
ISO Format Command	Yes	Yes
Format while Write Command	Yes	Yes
Restore/Save Command	Yes	Yes
Selectable Boot Drive	Yes	Yes
Drive ID Pins/Support	No	Yes
Enhanced SRB Support	Yes	Yes

Pin Changes on the 64-Pin Part:

- INVERT# is removed
- 4 NC's (no connects) are removed
- MFM, IDENT pins on the 82077SL have been changed to IDENT1 and IDENT0 respectively. The polarities of IDENT1 and IDENT0 enable the following modes of operation:

IDENT0	IDENT1	MODE
1	1	Standard AT/EISA
1	0	Illegal
0	1	PS/2
0	0	Model 30

- PLL0 pin, which allowed for H/W configuration of tape drive mode is no longer available. Tape mode can be configured via the TDR register.
- DENSEL, DRATE1, DRATE0 pins have been substituted by DRVDEN0, DRVDEN1. The Drive Specification command can be used to configure these pins for various requirements of drives available on the market.
- RDGATE has been added and can be used for diagnostics of the PLL.
- MEDID1, MEDID0 are new, they return media type information to the TDR register.
- DRVID1, DRVID0 return drive type information to the TDR register.
- SEL3V# selects between either 3.3V or 5V mode. Connecting the pin LOW selects 3.3V mode.
- 5 V_{SS} pins, 2 V_{CC} pins, 2 V_{SSP} pins, 1 V_{CCF} pin, and 1 A_{VCC} and 1 A_{VSS} pin.
- V_{CCF} can be used to interface a 5.0V or a 3.3V drive to the 28078 (when SEL3V# is low).

Pin Changes on the 44-Pin Part:

- FDS1# and FDME1# function as status outputs for PD and IDLE when the 44PD EN bit in the Power Down command is set.
- IDENT and MFM pins not required since the part is designed to be in AT mode
- DRV2 is used in PS/2 mode, and is not required for AT mode
- 3 V_{SS} pins, 2 V_{CC} pins, 1 A_{VSS} and 1 A_{VCC} pin

82078 Enhanced Features:

MEDID1 and MEDID0 provide media type information when connected to pins 17 and 21 of the disk interface on some drives. Usage: some drives use the holes on diskettes to identify the type of media and relay this information back to the 82078. BIOS can then easily access this information via the TDR register.

The DRVDEN0 and DRVDEN1 pins on the 82078 substitute the DENSEL, DRATE0, and DRATE1 pins on the 82077SL. These pins are configured via the Drive Specification Command alleviating the need for a hardware work around to accommodate various drives. By programming this command during the BIOS POST routine, the floppy disk controller will internally configure the correct values for DRVDEN0 and DRVDEN1 with corresponding pre-compensation and data rate tables enabled for the particular type of drive. Software resets will not clear the programmed parameters, only another Drive Specification Command or H/W reset will reset it to default values.

Power management performance can be improved by using the SAVE and RESTORE commands. The SAVE command provides 16 bytes of information regarding the status of the 82078. The RESTORE command, when used in conjunction with the SAVE command, allows the 82078 to power up from a 0V power down quickly. After issuing a RESTORE command, 16 bytes are written to the 82078; this information restores the controller to its original state.

The 82078 can also operate with a 48 MHz external oscillator. BIOS can set the CLK48 bit in the configure command during initialization.

To accommodate the new emerging tape drives, the 82078 now supports a 2 Mbps data rate. The Drive Specification command enables this mode. The 2 Mbps data rate is not supported if the 82078 is operated at 3.3V.

An External Architecture Specification (EAS) is available for the 82078. For more information, contact your local Intel Sales representative.

APPENDIX C
FLOPPY DISK CONTROLLER PROGRAM

```
/*************************************************************************
 *                                                                       *
 * Copyright (c) Intel Corporation 1992 - All Rights Reserved            *
 *                                                                       *
 * Name: Floppy Disk Example                                             *
 *                                                                       *
 * Version: 0.0                                                          *
 *                                                                       *
 * Author: Eric Auzas                                                    *
 *                                                                       *
 * Date: 10-19-92                                                        *
 *                                                                       *
 * Filename: MAIN.C                                                      *
 *                                                                       *
 * Language: Compiled using Microsoft C Version 7.0                      *
 *                                                                       *
 * Functional Description:A basic example of how to write random         *
 * numbers to a PC Compatible Floppy Disk File                           *
 *                                                                       *
 * Note: This is only an example and has several limitations. It will    *
 *       write the numbers to a file on a Floppy but it does not:        *
 *         - check to see if there is a floppy disk                      *
 *         - check if it is full                                         *
 *         - check on the floppy size                                    *
 *         - write to the second FAT (easily done)                       *
 *                                                                       *
 *************************************************************************/


#include "atkit.h"

void bios_setup(void);
void dos_setup(void);
void rd_data(void);


/* Timer2 PCB Address */
#define t2cnt 0xff60
#define t2con 0xff66
#define t2cmp 0xff62

/*************************************************************************
 *                                                                       *
 *                          Definitions                                  *
 *                                                                       *
 *************************************************************************/

/* 3.5 Media Descriptors */

#define FLD_144 0xF0
#define FLD_720 0xF9

/* FAT Descriptors */

#define  CLU_AVAIL 0x00
#define  CLU_RESL 0xFF0
#define  CLU_RESH 0xFF6
#define  CLU_BAD 0xFF7
#define  CLU_LASTL 0xFF8
#define  CLU_LASTH 0xFFF

/* ROOT Directory Descriptors */

#define  FILENAME 0x00
```

272339-6

```
#define  EXT 0x08
#define  FILE_ATT 0x0B
#define  RESERVED 0x0C
#define  CRE_TIME 0x16
#define  CRE_DATE 0x18
#define  CLU_START 0x1A
#define  FILE_SIZE 0x1C

/* First Byte Filename Descriptors */

#define  END 0x00
#define  CHAR_E5 0x05
#define  ALIAS 0x2E
#define  ERASED 0xE5

/* File Attribute Byte Descriptors */

#define  READ   0x00
#define  HIDDEN 0x01
#define  SYSTEM 0x02
#define  VOLUME 0x03
#define  DIRECTY 0x04
#define  ARCHIVE 0x05

/*************************************************************************
 *                                                                       *
 *                          Boot Block Definitions                       *
 *                                                                       *
 *************************************************************************/

struct boot_block
{
char jump[3];
char oem[8];
int byte_sec;
char sec_clu;
int res_sec;
char num_fat;
int num_rdir;
int sec_vol;
char media;
int sec_fat;
int sec_trk;
char num_head;
char dos40[34];
char bootstrap[451];
};

struct boot_block far boot,far *boot_ptr;

/*************************************************************************
 *                                                                       *
 *                            FAT Block Definitions                      *
 *                                                                       *
 *************************************************************************/

struct fat_block
{
char index[4608];
char end;
};

struct fat_block far fat,far *fat_ptr;
```

272339-7

```
/*************************************************************************
 *                                                                       *
 *                    Directory Entry Definitions                        *
 *                                                                       *
 *************************************************************************/

struct dir_entry_block
{
char filename[8];
char ext[3];
char file_atrb;
char reserved[10];
int time;
int date;
int start_clu;
unsigned long file_size;
};

struct dir_entry_block far dir[224], far *dir_ptr;

char far *p;
unsigned buf_seg;

/*************************************************************************
 *                                                                       *
 *                     Random Number Storage                             *
 *                                                                       *
 *************************************************************************/


struct data_buf
{
char index[4];
char del1;
char info[4];
char cr;
char lf;
};
struct data_buf far data[200], far *data_ptr;
int data_cnt;
int long data_size;

/*************************************************************************
 *                                                                       *
 *                   Generate Random Numbers 0 to 9999                   *
 *                                                                       *
 *************************************************************************/

void rd_data()
 {
  unsigned num,digit,whole;
  int i;
  data_ptr=data;
  data_cnt=0;
  outpw(t2con,0x4000);   /* Timer off */
  outpw(t2cmp,0x270f);   /* Count 0 to 9999 */
  outpw(t2con,0xc001);   /* Timer on, no interrupts, continuous mode */
  do
   {
    num=data_cnt;
    digit=1000;
    for (i=3; i>=0 ; i=i-1)            /* Convert Index into ASCII */
     {
      whole=num/digit;
```

272339-8

```
      data[data_cnt].index[3-i]=whole+0x030;
      num=num-(whole*digit);
      digit=digit / 10;
     }
    data_ptr[data_cnt].deli1=',';      /* Separate Index and Random Number */
                                       /* with a comma as a delimiter */
    num=inpw(t2cnt);
    digit=1000;
    for (i=3; i>=0; i=i-1)             /* Convert Number into ASCII */
     {
      whole=num/digit;
      data[data_cnt].info[3-i]=whole+0x030;
      num=num-(whole*digit);
      digit=digit / 10;
     }
    data_ptr[data_cnt].cr=0x000D;      /* Add a carriage return and */
    data_ptr[data_cnt].lf=0x000A;      /* a linefeed at the end     */
    data_cnt=data_cnt+1;
   }
  while(data_cnt <= 200 );
  data_size = data_cnt * 11;
  outpw(t2con,0x4001);                 /* Turn off Timer 2 */
 }

/*************************************************************************
 *                                                                       *
 *                        Read A Sector                                  *
 *                                                                       *
 *************************************************************************/

void sec_read(char num_sec, char cyl, char sec,char head, char drive, unsigned
buf_seg,unsigned buf_off)
{
__asm
{
mov ch,cyl
mov cl,sec
mov DH,head
mov DL,drive
mov AL,num_sec
mov AH,02H
mov BX,buf_seg
mov ES,BX
mov BX,buf_off
int 50
}
}

/*************************************************************************
 *                                                                       *
 *                        Write A Sector                                 *
 *                                                                       *
 *************************************************************************/

void sec_write(char num_sec, char cyl, char sec,char head, char drive,unsigned
buf_seg,unsigned buf_off)
{
__asm
{
mov ch,cyl
mov cl,sec
mov DH,head
mov DL,drive
mov AL,num_sec
```

272339-9

```
mov AH,03H
mov BX,buf_seg
mov ES,BX
mov BX,buf_off
int 50
}
}

/*************************************************************************
 *                                                                       *
 *                          Search Directory for Openings                *
 *                                                                       *
 *************************************************************************/

unsigned search_dir(void)
{
 unsigned flag,index;
 flag=0;
 index=0;
 do
  {
   if ((dir[index].filename[0] == 00) | (dir[index].filename[0] == 0x00E5))
     flag=1;
   index=index+1;
   if (index==0x0600)
     flag=1;
  }
 while (flag==0);
 return(index-1);
}

/*************************************************************************
 *                                                                       *
 *                Encode Fat with File Sector Locations                  *
 *                                                                       *
 *       This procedure is recursive and looks in the FAT for open       *
 *       sector locations for the file to be written.                    *
 *       This routine assumes a 12 bit FAT is being used.                *
 *       Three bytes represent two FAT entries. The three bytes need to  *
 *       be decoded to determine the two FAT entries.                    *
 *       There is one sector per cluster. This routine does not check    *
 *************************************************************************/

unsigned found[2048], cur_dir;
void fat_encode(unsigned sec_count,unsigned done,signed count)

 {
  unsigned cur_clus,clus1,clus2;
  unsigned byte1, byte2, byte3;

  cur_clus = (sec_count * 3)/2;        /* Cluster = Sector * 1.5 */
  byte1 = fat.index[cur_clus] & 0x00ff;
  byte2 = fat.index[cur_clus+1] & 0x00ff;
  byte3 = fat.index[cur_clus+2] & 0x00ff;
  clus1 = (byte2 & 0x00f) * 0x0100 + byte1; /* Decode Cluster entry from bytes */
  clus2 = (byte3 * 0x010) + (byte2 & 0x0f0) / 0x010;

  if ((clus1==0) && (clus2==0))        /* Are the both cluster open? */
   {
    found[count] = sec_count;          /* Save the 1st sector number */
    count++;
    found[count] = sec_count+1;        /* Save the 2nd sector number */
    count++;
    if (count < done)                  /* Recursive call until empty cluster */
```

272339-10

```
     {
      sec_count=sec_count+2;
      fat_encode(sec_count,done,count);/* entries have been found for each   */
     }                                 /* sector.                            */
    if (count > done-1)                /* last cluster */
      clus2 = 0x0fff;
    else
      clus2 = found[count];            /* Save cluster 2 pointer to next cluster*/
    clus1 = found[count-1];            /* Save cluster 1 pointer to next cluster*/

   }
  else if (clus1==0)                   /* Check if cluster 1 is free */
   {
    found[count] = sec_count;          /* Array FOUND keeps track of the */
    count++;                           /* sectors free.                  */
    if (count < done)                  /* Recursive call until empty cluster */
     {
      sec_count = sec_count + 2;
      fat_encode(sec_count,done,count);/* entries have been found for each   */
                                /* sector.                            */
     }
    if (count>done)                    /* last cluster */
      clus1 = 0x0fff;
    else
      clus1 = found[count];            /* Save cluster 1 pointer to next cluster */
   }

  else if (clus2==0)                   /* Check if cluster 2 is free */
   {
    found[count] = sec_count+1;        /* Array FOUND keeps track of the */
    count++;                           /* sectors free                   */
    if (count < done)                  /* Recursive call until empty cluster */
     {
      sec_count = sec_count + 2;
      fat_encode(sec_count,done,count);/* entries have been found for each   */
                                /* sector.                            */
     }
    if (count > done-1)                /* last cluster */
      clus2 = 0x0fff;
    else                               /* Save cluster 2 pointer to next cluster*/
      clus2 = found[count];
   }
  else                                 /* no empty clusters go to next two */
   {
    if (count < done)
     {
      sec_count = sec_count + 2;
      fat_encode(sec_count,done,count);
     }
   }
  if (count==1)                        /* Save first cluster to directory entry*/
    dir[cur_dir].start_clu = found[count];

  byte1 = clus1 & 0x0ff;               /* save clusters to FAT table */
  byte2 = ((clus1 & 0xf00) >> 8) | ((clus2 & 0x00f) << 4);
  byte3 = (clus2 & 0x0ff0) >> 4;
  fat.index[cur_clus] = byte1;
  fat.index[cur_clus+1] = byte2;
  fat.index[cur_clus+2] = byte3;
 }


/*************************************************************************
 *                                                                       *
```

272339-11

```
 *             Define an example File Directory Entry                    *
 *             to write the random numbers to.                           *
 *                                                                       *
 *************************************************************************/

void dir_encode()
{
dir[cur_dir].filename[0] = 'F';
dir[cur_dir].filename[1] = 'D';
dir[cur_dir].filename[2] = 'C';
dir[cur_dir].filename[3] = ' ';
dir[cur_dir].filename[4] = ' ';
dir[cur_dir].filename[5] = ' ';
dir[cur_dir].filename[6] = ' ';
dir[cur_dir].filename[7] = ' ';
dir[cur_dir].ext[0]='T';
dir[cur_dir].ext[1]='X';
dir[cur_dir].ext[2]='T';
dir[cur_dir].file_atrb=0x0020;
dir[cur_dir].file_size=data_size;
dir[cur_dir].time=1500;
dir[cur_dir].date=9212;
dir[cur_dir].start_clu = found[0];
}

/*************************************************************************
 *                                                                       *
 *                    Setup BIOS for DOS file access                     *
 *                                                                       *
 *************************************************************************/


void bios_setup()
{
TIMER_LONG=0;
TIMER_LONG=1;
dsetup();
fdisk_setup();
}

/*************************************************************************
 *                                                                       *
 *                      Write DOS File                                   *
 *                                                                       *
 *************************************************************************/

void dos_setup()

{

int i,j;
unsigned int cyl,sectors,flag;
unsigned buf_size,no_sec;
unsigned int rem1,abs_sec,cal_sec;

char fat1_sec, fat2_sec;
char dir_sec;
char head,drive;
int dir_len,data_len,fat_len;
boot_ptr = &boot;
fat_ptr = &fat;
dir_ptr= &dir[0];                       /* Located backwards in memory */

/* Load the Boot Sector */
```

272339-12

```
sec_read(01,00,01,00,00,get_seg(boot_ptr),&boot);

head=0;
cyl=0;
drive=0;
fat1_sec=2;                              /* 1st FAT located at sector 2 */
fat2_sec=fat1_sec+(boot_ptr->sec_fat);  /* 2nd FAT located at after 1st FAT */
fat_len=fat2_sec+(boot_ptr->sec_fat)-1; /* Calculate last sector of 2nd FAT */
/* Calculate location of first sector of the Directory Table */
dir_sec=fat1_sec+((boot_ptr->sec_fat)*(boot_ptr->num_fat));
if (dir_sec > 18)
 {
  if (head==0)
    head=1;
  else
    {
    head=0;
    cyl=cyl+1;
    }
  dir_sec=dir_sec-18;
 }
dir_len=(0x0020*(boot_ptr->num_rdir))/0x200;    /* Calculate Directory Length */
data_len=dir_len+fat_len;                       /* Calculate first sector of Data area */

/* Read FAT */
sec_read(boot.sec_fat,00,fat1_sec,00,00,get_seg(fat_ptr),&fat);

/* Read Root Directory */
do
{
sectors=dir_len;                         /*Set number of sectors to directory size*/
if ((dir_len+dir_sec) > 18)
  {
  sectors=19-dir_sec;
  flag=1;
  }
sec_read(sectors,cyl,dir_sec,head,drive,get_seg(dir_ptr),&dir);

if (flag == 1)
  {
   dir_len=dir_len-sectors;
   dir_ptr=&dir[224-sectors];
   cyl=cyl+1;
   flag=0;
   dir_sec=1;
   }
else
   dir_len=0;

}
while (dir_len > 0);

no_sec = (data_size / 512);              /*Calculate number of sectors needed*/
                                 /*to store data */
if (data_size % 512 != 0)                /*Add one if there is a remainder */
 {
  if ((data_size-(no_sec * 512) > 0) || (no_sec < 0))
    no_sec=no_sec+1;
 }
cur_dir=search_dir();                    /* Search for open directory entry/*
fat_encode(2,no_sec,0);                  /* Place file in FAT */
dir_encode();                            /* Create File directory entry */
```

272339-13

```
/* Write Data to Disk */

for (i=0;i<no_sec;i=i+1)

{
abs_sec=(found[i]-2)+data_len;          /* Determine absolute sector */
cyl=abs_sec/36;                         /* Determine if cylinder 0 or 1 */
rem1=(abs_sec-(36*cyl));                /* Get remainder */
head=rem1/18;                           /* If remainder > 18 then head 1 */
cal_sec=(rem1-(head*18))+1;             /* Get sector number */
sec_write(01,cyl,cal_sec,head,00,get_seg(data),((char*)(&data))+(512*i));
}

/* Write  FAT AND DIRECTORY to Disk */
cyl=0;
sec_write(boot.sec_fat,00,fat1_sec,00,00,get_seg(fat_ptr),&fat);
head=1;
sec_write(sectors,cyl,dir_sec,head,drive,get_seg(dir_ptr),&dir);

}

main()

{
 rd_data();                             /* Get random numbers            */
 bios_setup();                          /* Initialize Floppy BIOS routine*/
 dos_setup();                           /* Write random numbers to       */
                              /* a file on the Floppy          */
 wait_loop:
 goto wait_loop;
}
```

272339-14

```
/*************************************************************************
*                                                                        *
* Copyright (c) FOSCO 1988, 1989, 1991 - All Rights Reserved             *
*                                                                        *
* Module Name: AT Bios enhanced floppy disk driver                       *
*                                                                        *
* Version: 2.00                                                          *
*                                                                        *
* Author: JOHN FOSCO(Annabooks)                                          *
*                                                                        *
* Modifications done by: BRENDAN RUIZ(Intel), ERIC AUZAS(Intel)          *
*                                                                        *
* Date: 10-19-92                                                         *
*                                                                        *
* Filename: ATDISK.C                                                     *
*                                                                        *
* Language: Compliled using Microsoft C Version 7.0                      *
*                                                                        *
* Functional Description: Floppy Disk Driver modified for the 80C186     *
*                                                                        *
* Version History:                                                       *
* 1.01-1.03a                                                             *
*   AT BIOS                                                              *
* 2.00  10-19-92                                                         *
*  Modified and deleted un-necessary routines to run with the            *
*  Intel 80C186XL/EA Evaluation Board & 82077AA Floppy Disk Controller   *
*  Board.                                                                *
*                                                                        *
* Note:  This software must be registered by Annabooks Inc.              *
*        Annabooks can be reached at 1-800-462-1042                      *
*************************************************************************/

/* I N C L U D E   F I L E S */

#include "kit.h"

/* F U N C T I O N   P R O T O T Y P E S */

void set_vector(int_number,seg,off);
unsigned acquire_scratch_block(unsigned id,unsigned block_size);
void dsetup(void);
void fdisk_setup(void);
void interrupt far fdisk_io(void);
void interrupt far fdisk_isr(void);
void interrupt timer_int(void);
unsigned send_fdc(unsigned char);
void send_rate(unsigned);
unsigned retry(unsigned);
unsigned med_change(unsigned);
unsigned char calc_sectors(unsigned);
unsigned GetParm(unsigned,unsigned char);
unsigned wait_int(void);
unsigned recal(unsigned);
unsigned seek(unsigned);
void wait_for_head(unsigned);
unsigned read_id(unsigned);
unsigned get_fdc_status(unsigned);
unsigned char read_dskchng(unsigned);
unsigned results(void);
unsigned chk_stat_2(void);
void send_specify_command(unsigned);
void FDC_reset(unsigned);
void delay_call(unsigned,unsigned);
unsigned dma_setup(unsigned,unsigned,unsigned);
```

272339-15

```
void purge_fdc(void);
unsigned fdc_init(unsigned);
void motor_on(unsigned);

/* G L O B A L   C O N S T A N T S */

extern _fdisk_io;
extern _fdisk_isr;

/* L O C A L   D E F I N I T I O N S */

#define FDC_STATUS   0x42
#define DISK_ID    0x90 /* hold drive/media type  */
#define LAST_TRACK 0x94 /* 94-97 holds last track number */
#define RATE_500 0x00
#define RATE_300 0x01
#define RATE_250 0x02
#define RATE_1000       0x03
#define INT_FLAG BIT7
#define MOTOR_WAIT 0x25
#define MOTOR_OFF 0X0C
#define BAD_CMD  0x01
#define BAD_ADDR_MARK 0x02
#define WRITE_PROTECT 0x03
#define RECORD_NOT_FND 0x04
#define MEDIA_CHANGE 0x06
#define BAD_DMA  0x08
#define DMA_BOUNDARY 0x09
#define MED_NOT_FND 0x0c
#define BAD_CRC  0x10
#define BAD_FDC  0x20
#define BAD_SEEK 0x40
#define TIME_OUT 0x80

/* function code definitions */

#define RESET  0x00
#define READ_STATUS 0x01
#define READ_SECTORS 0x02
#define WRITE_SECTORS 0x03
#define VERIFY_SECTORS 0x04
#define FORMAT_TRACK 0x05
#define DISK_PARMS 0x08
#define DISK_TYPE 0x15
#define DISK_CHANGE 0x16
#define FORMAT_SET 0x17
#define SET_MEDIA 0x18
#define BAD_FUNCTION 0x19
#define TRY  0xff

/* FDC port definitions */

#define DOR_PORT   0x604
#define MSR_PORT  0x608
#define DATA_PORT  0x60A
#define DIR_PORT  0x60E
#define DRR_PORT  0x60E
#define DACK_PORT 0x61A
#define TC_PORT 0X63A
#define RQM  BIT7
#define DIO  BIT6
#define BUSY  BIT4
#define DSKCHANGE_BIT BIT7
```

272339-16

```
/* Interrupt Controller Registers */
#define EOI 0xff22    /* end of interrupt */
#define I0CON 0xff38   /* interrupt 0 control */
#define IMASK 0xff28  /* interrupt mask */
#define INT0_EOI 0x000C /*INT 0 End of Interrupt Command */
#define FDC_INT 0x0032 /* Non reserved 186 interrupt */

/* Interrupt Vector info */
#define TMR2_INT 0x13   /* Timer 2 Interrupt */
#define ISR 0x0c        /* Hardware Service Routine (for INT0) */
#define DISK_IO 0x32    /* Disk I/O Routine */
#define DISK_TABLE 0x1E /* Disk Parameters Table  */

/* Timer 2 Port definitions */
#define TMR2_CON 0xff66
#define TMR2_CMP 0xFF62
#define TMR2_COUNT 0xff60
#define TCUCON 0xff32
#define TMR2_EOI 0x0008         /* Timer 2 End of Interrupt Command */
#define TMR2_SET 0xE001         /* Timer enabled, continous mode, interrupts enabled */
#define TMR2_CNT 0x0FA0         /* 50us Tick at 20MHz

/* DMA channel 0 port definitions */
#define D0SRCL   0xffc0
#define D0SRCH   0xffc2
#define D0DSTL   0xffc4
#define D0DSTH   0xffc6
#define D0TC   0xffc8
#define D0CON   0xffca

/* DMA literal definitions */
#define DMA_ON  0x02
#define DMA_OFF  0x06
#define DMA_RX_MODE 0x46
#define DMA_TX_MODE 0x4a
#define DMA_VRFY_MODE 0x42
#define TX_DIR  0x01
#define RX_DIR  0x00

/* FDC commands */
#define FDC_READ 0xe6
#define FDC_WRITE 0xc5
#define FDC_FORMAT 0x4d
#define FDC_READID 0x4a

/*********************************************************************
 *             These are the floppy disk parameter tables            *
*                                                                     *
 *                                                                   *
*                                                                     *
 * The tables are organized as an Array of up to 8 Drive types, each  *
 * supporting up to 8 media types. Each item is 16 bytes long        *
 *********************************************************************/

/* DISK TABLE indexes */

#define DT_SPEC1 0      // specify command 1
#define DT_SPEC2 1      // specify command 2
#define DT_OFF_TIM 2    // motor off count
#define DT_BYT_SEC 3    // bytes/sector
#define DT_SEC_TRK 4    // sectors/track
#define DT_GAP 5        // gap
#define DT_DTL 6        // dtl
#define DT_GAP3 7       // gap 3 for format command
```

272339-17

```
#define DT_FIL_BYT 8    // fill byte for format command
#define DT_HD_TIM 9     // head settle time
#define DT_STR_TIM 10   // motor start time
#define DT_MAX_TRK 11   // max # of tracks for drive
#define DT_RATE 12      // data rate
#define DT_TYPE         // drive and media type (not used)
#define DT_STEP 14      // double step flag

#define drive_established 0x08
#define drive_field 0x07
#define drive_none  00
#define drive_360  01
#define drive_12  02
#define drive_720  03
#define drive_14  04
#define drive_28  05

#define media_established 0x80
#define media_field 0x70
#define media_none  0x00
#define media_360  0x10
#define media_12  0x20
#define media_720  0x30
#define media_14  0x40
#define media_28  0x50
// media definitions for transition table

#define m_none  0x0
#define m_360  0x1
#define m_12  0x2
#define m_720  0x3
#define m_14  0x4
#define m_28  0x5

#define m_wait 0x25

/* L O C A L   C O N S T A N T S */

const unsigned char transition_table[8][4]= {
// This table covers the sequence of media type to try for
// each drive type in establishing the media in the drive.
// The media is defaulted to the first item in this table
// for a particular drive type. When we attempt retries, we
// step through the possible media types until we come
// to "none" marking the end of the table.

/* drive type 0 (None) */ {m_none,m_none,m_none,m_none},
/* drive type 1 ( 360) */ {m_360, m_none,m_none,m_none},
/* drive type 2 ( 1.2) */ {m_12,  m_360, m_none,m_none},
/* drive type 3 ( 720) */ {m_720, m_none,m_none,m_none},
/* drive type 4 (1.44) */ {m_14,  m_720, m_none,m_none},
/* drive type 5 (2.88) */ {m_28,  m_14,  m_720, m_none},
/* drive type 6 unused */ {m_none,m_none,m_none,m_none},
/* drive type 7 unused */ {m_none,m_none,m_none,m_none},
};

const unsigned char fdisk_table[8][8][16]= {
/* this entry is for compatibility with the XT disk driver */
{
 /* ---- drive type 0 = None (Default to 360) ----*/

 {0x0af,2,m_wait,2,9,0x2a,-1,0x50,0x0f6,15,8,39,RATE_250,drive_360|media_360,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
```

272339-18

```
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 },

 {
  /* ---- drive type 1 = 360 ----*/

  /* 0 = no media in 360 kb drive */
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

  /* 1 = 360 kb media in 360 kb drive */
  {0x0af,2,m_wait,2,9,0x2a,-1,0x50,0x0f6,15,8,39,RATE_250,drive_360|media_360,0,0},

  /* 2-7 not used */
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 },


 {
  /* ---- drive type 2 = 1.2 ------*/

  /* 0 = no media in 1.2 mb drive */
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

  /* 1 = 360 kb media in 1.2 mb drive */
  {0xaf,2,m_wait,2,9,0x2a,-1,0x50,0x0f6,15,8,39,RATE_300 ,drive_12|media_360,1,0},

  /* 2 = 1.2 mb media in 1.2 mb drive */
  {0xaf,2,m_wait,2,15,0x1b,-1,0x54,0x0f6,15,8,79,RATE_500 ,drive_12|media_12,0,0},

  /* 3-7 = not used */
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 },


 {
  /* ---- drive type 3 = 720 ------ */

  /* 0 = no media in 720 kb drive */
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

  /* 1-2 = not used */
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

  /* 3 = 720 kb media in 720 kb drive */
  {0x0af,2,m_wait,2,9,0x2a,-1,0x50,0x0f6,15,8,79,RATE_250 ,drive_720|media_720,0,0},

  /* 4-7 = not used */
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
```

272339-19

```
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
},

{
 /* ---- drive type 4 = 1.44 ------*/

 /* 0 = no media in 1.44 mb drive */
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

 /* 1-2 =  not used */
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

 /* 3 =  720 kb media in 1.44 mb drive */
 {0xaf,2,m_wait,2,9,0x2a,-1,0x50,0x0f6,15,8,79,RATE_250 ,drive_14|media_720,0,0},

 /* 4 = 1.44 kb media in 1.44 mb drive */
 {0xaf,2,m_wait,2,18,0x1b,-1,0x6c,0x0f6,15,8,79,RATE_500 ,drive_14|media_14,0,0},

 /* 5-7 = not used */
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
},

{

 /*---- drive type 5 = 2.88 ------*/

 // The 2.88 Meg drives have not been tested.
 // These tables are included as a help to integrating 2.88 drives into your system.
 // The correct parameters must be determined in conjunction with the drive manufacturers
 // specifications.

 /* 0 = no media in 2.88 mb drive */
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

 /* 1-2 not used */
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

 /* 3 = 720 kb media in 2.88 mb drive */
 {0xaf,2,m_wait,2,9,0x2a,-1,0x50,0x0f6,15,8,79,RATE_250 ,drive_28|media_720,0,0},

 /* 4 = 1.44 kb media in 2.88 mb drive */
 {0xaf,2,m_wait,2,18,0x1b,-1,0x6c,0x0f6,15,8,79,RATE_500 ,drive_28|media_14,0,0},

 /* 5 = 2.88 kb media in 2.88 mb drive */
 {0xaf,2,m_wait,2,36,0x1b,-1,0x53,0x0f6,15,8,79,RATE_1000,drive_28|media_28,0,0},

 /* 6-7 not used */
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
},

{
 /*---- drive type 6 = reserved ------*/

 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
```

272339-20

```
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
},

{
 /*---- drive type 7 = reserved ------*/

 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
 {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
},

};

/************************************************************************
 *                                                                      *
 *          FOSCO Enhanced Floppy Disk Driver                           *
 *                                                                      *
 *          Copyright FOSCO 1988, 1989
*
 *                                                                      *
 ************************************************************************/

/*
This driver uses a drive/media id byte as follows:

bit 7 0/1 = media not established / established

bit 6-4 media type
000 = no established media
001 = 360 (40/80 track)
010 = 1.2 (80 track)
011 = 720 (80 track)
100 = 1.4 (80 track)
101 = 2.8 (80 track)
110 - 111 reserved

bit 3 = 0/1 = drive not established / established

bit 2-0 Drive type
000 no established drive type
001 = 360 (40 track)
010 = 1.2 (80 track)
011 = 720 (80 track)
100 = 1.4 (80 track)
101 = 2.8 (80 track)
110 - 111 reserved

the drive/media bytes are located at 40:90-93 for up to four drives.*/

/************************************************************************
 *                                                                      *
 *                          Start of Code                               *
 *                                                                      *
 ************************************************************************/

unsigned int r_w_flag;
unsigned v_es,last_address;
char last_byte;
```

272339-21

```
variables
unsigned char fd_drive_type;
unsigned char fd_state;
unsigned char fd_fdc_command;
unsigned char fd_dma_command;
unsigned char fd_sector_track;
unsigned char fd_media_type;
unsigned char fd_fcode;
unsigned char fd_drive;
unsigned char fd_nr_sectors;
unsigned char fd_head;
unsigned char fd_sector;
unsigned char fd_track;
unsigned fd_max_track;
unsigned fd_max_sectors;
unsigned fd_i;
unsigned fd_try_count;
unsigned fd_media_index;
unsigned char end_track,end_head,end_sector,num_sectors,sectors_track;
unsigned dma_address, dma_control;
unsigned nibble, word,  length;
unsigned char error_byte,time_out_flag;
unsigned long seg_check;
unsigned *last_address;
end_variables fdisk_regs;

/*************************************************************************
 *                                                                       *
 *                        Set Interrupt Vector                           *
 *                                                                       *
 *************************************************************************/

void set_vector(int_number,seg,off)
{
 _disable();
 poke(0x00,int_number * 4,off);
 poke(0x00,(int_number * 4) + 2,seg);
 _enable();
}

#define swapped 1       // in myblock->status, this tells if stack is swapped

extern unsigned start_block;
extern unsigned current_open;
extern unsigned end_block;

variables
end_variables poolregs;

/*************************************************************************
 *                                                                       *
 *                        Acquire Scratch Block                          *
 *                                                                       *
 *          Block size is passed as number of bytes needed.              *
 *                                                                       *
 *************************************************************************/

unsigned acquire_scratch_block(unsigned id,unsigned block_size)
{
 poolregs *block_ptr;
 setds_system_segment();
 _disable();
 if ((current_open + block_size) < end_block)
```

272339-22

```
 {
  block_ptr = current_open;
  current_open += block_size;
  block_ptr->length_tag = block_size;
  block_ptr -> user_id = id;

  _enable();
  return(block_ptr);
 }
 // acquire the start block so we can flag system error
 current_open = start_block;
 _enable();
 return(current_open); // this is an error condition
}

/*************************************************************************
 *                                                                       *
 *                        Release Block                                  *
 *                                                                       *
 *************************************************************************/

void release_block(poolregs *block_ptr)
{
 _disable();
 clear_block(block_ptr,block_ptr -> length_tag);
 current_open = block_ptr;
 _enable();
}

/*************************************************************************
 *                                                                       *
 *     Set up interrupt controller, timer 2 and timer interrupt          *
 *                                                                       *
 *************************************************************************/

void dsetup()
 {
  _disable();
  outpw(I0CON,0x0009);   /*INT0 disabled, edge triggered, priority level 1 */
  outpw(TCUCON,0x0008); /* TMR2 interrupt disabled, priority level 0 */
  outpw(IMASK,0x00fd);   /* mask all interrupts */
  /* unmask interrupts one at a time when each one is ready */
  /* --------------Timer_Setup--------------- */
  /* set up timer interrupt vector */
  link_interrupt(TMR2_INT,timer_int);

  /* initialize timer 2 */
  outpw(TMR2_COUNT,0x0000);
  outpw(TMR2_CMP, TMR2_CNT);                    /* 50 us Clock Tick */
  outpw(TMR2_CON, TMR2_SET);                    /*timer enabled, continous mode,
interrupts enabled */
  TIMER_STEP=0;
  outpw(TCUCON,inpw(TCUCON) & ~BIT3);           /* unmask interrupt  */
  _enable();
}

/*************************************************************************
 *                                                                       *
 *     Timer Interrupt Service Routine                                   *
 *                                                                       *
 *************************************************************************/

void interrupt far timer_int(void)
 {
```

272339-23

```
  _disable();
  DELAY_LONG=DELAY_LONG+1;
  TIMER_STEP=TIMER_STEP+1;
  if (TIMER_STEP==1000)
   {                    /*50 millisecond tick*/
    TIMER_LONG=TIMER_LONG+1;
    TIMER_STEP=0;
   }
  __asm
   {
    mov     dx,TMR2_CON
    mov     ax,TMR2_SET
    out     dx,ax
    mov     dx,EOI
    mov     ax,TMR2_EOI
    out     dx,ax
   }
  _enable();
 }

/************************************************************************
 *                                                                      *
 *     Set up the Floppy Disk Controller                                *
 *                                                                      *
 ************************************************************************/

void fdisk_setup()
{
 unsigned int seg,off;
 unsigned far *p;

 fdisk_regs *myblock;
 move_system_segment();
 myblock = acquire_block(FLOPPY);
 myblock->fd_drive_type= 0x0004;  /*From table above */
 myblock->fd_media_type= 0x0004;
 link_interrupt(DISK_IO,fdisk_io);
 link_interrupt(ISR,fdisk_isr);
 p=&fdisk_table;
 seg=get_seg(p);
 set_vector(DISK_TABLE,seg,&fdisk_table[0][0][0]);
 outpw(IMASK,inpw(IMASK) & ~BIT0);    /* unmask interrupt */
 outpw(IMASK,inpw(IMASK) & ~BIT4);    /* unmask INT0  */
 SEEK_STATUS = 0;   /* clear seek status */
 MOTOR_COUNT = 0;   /* clear motor count */
 MOTOR_STATUS = 0;  /* clear motor status */
 DISK_STATUS = 0;   /* clear disk status */
 pokeb40(DISK_ID+0,0);   /* clear drive 0 state */
 pokeb40(DISK_ID+1,0);   /* clear drive 1 state */
 pokeb40(DISK_ID+2,0);   /* clear drive 2 state */
 pokeb40(DISK_ID+3,0);   /* clear drive 3 state */

 /* drive type = 1.44 MByte        !!! Change if using different drive size */
 pokeb40(DISK_ID+0,0xCC);
 pokeb40(DISK_ID+1,0x00);       /* only 1 drive in system */
 pokeb40(DISK_ID+2,0x00);
 pokeb40(DISK_ID+3,0x00);
 MOTOR_COUNT = GetParm(myblock,2);      /* set motor count */
 myblock -> dx = 0x0000;
 myblock -> ax = 0x0000;  /* reset the FDC */
 /* sys_int();   */
 sys_int(FDC_INT,myblock);
 release_block(myblock);
}
```

272339-24

```
/*************************************************************************
 *                                                                       *
 *     Main Programs                                                     *
 *                                                                       *
 *************************************************************************/

/* handle the floppy disk function service call */

void interrupt far fdisk_io(interrupt_registers)
{
 fdisk_regs *myblock;
 myblock = acquire_block(FLOPPY);
 myblock->fd_fcode = ax >> 8;
 myblock->fd_drive = dx & 0x00ff;
 myblock->fd_nr_sectors = ax & 0x00ff;
 myblock->fd_head = dx >> 8;
 myblock->fd_sector = cx & 0x00ff;
 myblock->fd_track = cx >> 8;
 v_es = es;
 flags &= ~0x0001;  /* clear the carry flag for returns */
 DISK_STATUS = 0;  /* clear status */
 if (((ax >> 8) != 0) && ((dx & 0x00ff) > 3))
  { // bad function code because of drive number
   DISK_STATUS = BAD_CMD; ax = (BAD_CMD << 8) | (ax & 0xff);
   flags |= 1;  /* set return error flag */
  }
 else
  { /* make sure any residual status is unloaded */
   purge_fdc();
   /* set the data rate register to a default value */
   outp(DRR_PORT,RATE_500);
   switch (myblock->fd_fcode)
    {/*****************************************
      *   Reset Disk Controller               *
      *****************************************/

     case RESET :
       FDC_reset(myblock);
       ax = (DISK_STATUS << 8) | (ax & 0xff);
       if ((ax & 0xff00) != 0) flags |= 1;
     break;

     /*****************************************
      *   Read Status                         *
      *****************************************/

     case READ_STATUS: /* read the disk status */
       ax = DISK_STATUS << 8;
       if ((ax & 0xff00) != 0) flags |= 1;
     break;

     /*****************************************
      *   Read Sectors                        *
      *****************************************/

     case READ_SECTORS: /* read sectors */
       MOTOR_STATUS &= ~INT_FLAG;
       myblock->fd_fdc_command = FDC_READ;
       myblock->fd_dma_command = DMA_RX_MODE;
       r_w_flag  = 0;
     goto read_write_verify;

     /*****************************************
```

272339-25

```
 *   Write Sectors                          *
 ******************************************/

case WRITE_SECTORS:
  MOTOR_STATUS |= 0x80;
  myblock->fd_fdc_command = FDC_WRITE;
  myblock->fd_dma_command = DMA_TX_MODE;
  r_w_flag = 1;
goto read_write_verify;

/******************************************
 *   Verify Sectors                         *
 ******************************************/

case VERIFY_SECTORS:
  MOTOR_STATUS &= ~INT_FLAG;
  myblock->fd_fdc_command = FDC_READ;
  myblock->fd_dma_command = DMA_VRFY_MODE;
  r_w_flag  = 1;
goto read_write_verify;

/******************************************
 *   Read/Write/Verify                      *
 ******************************************/

read_write_verify:
  // if a media change sensed, then de-establish media
  if (med_change(myblock) == error)
 {
  andb40(DISK_ID+myblock->fd_drive,0x0f);
  myblock->fd_media_index = 0;
 }
  myblock->fd_try_count = 3;
  do  // this is the major loop, try the major operation 3 times
 {
  // if media not established, then set media type = drive type
  // set media_index = 0, set media type by index
  if ((peekb40(DISK_ID+myblock->fd_drive) & media_established) == 0)
   {
    // set the media index for the first possible type of media
    myblock->fd_drive_type = peekb40(DISK_ID+myblock->fd_drive) & 0x07;
    //reset the DISK_ID with the first media type to try
    pokeb40(DISK_ID+myblock->fd_drive,(peekb40(DISK_ID+myblock->fd_drive) &
    ~media_field) |
    (peekbcs(&transition_table[myblock->fd_drive_type][0]) << 4));
    // set the media type according to the drive type [index = 0]
    myblock->fd_media_type = (peekb40(DISK_ID+myblock->fd_drive) >> 4) & 0x07;
   }
  do  // this is the minor loop do until we run out of retrys
   {
    purge_fdc();     /* make sure any residual status is unloaded */
    // set up drive and media indexes before doing the dma setup,
    // which needs to know how many bytes per sector for calcing xfr length
    myblock->fd_drive_type = peekb40(DISK_ID+myblock->fd_drive) & 0x07;
    myblock->fd_media_type = (peekb40(DISK_ID+myblock->fd_drive) >> 4) & 0x07;
    if (dma_setup(myblock,es,bx) == ok)
     {
      /* send the specify command */
      send_fdc(3);
      send_fdc(GetParm(myblock,0));
      send_fdc(GetParm(myblock,1));
      send_rate(myblock);
      if (fdc_init(myblock) == ok)
       {
```

272339-26

```
            if (send_fdc(myblock->fd_track) == ok)
           {
            if (send_fdc(myblock->fd_head) == ok)
             {
              if (send_fdc(myblock->fd_sector) == ok)
               {
                if (send_fdc(GetParm(myblock,3)) == ok)
                 {
                  if (send_fdc(GetParm(myblock,4)) == ok)
                 {
                  if (send_fdc(GetParm(myblock,5)) == ok)
                   {
                    if (send_fdc(GetParm(myblock,6)) == ok)
                     __asm
                     {
                      push ax
                      push dx
                      push bx
                      out dx, al
                      mov dx, D0CON      ; enable DMA
                      cli
                bloop:  in al,dx           ; wait for STRT bit
                      test al,2          ; telling when Terminal
                      jnz bloop          ; has been reached
                      mov bx,r_w_flag
                      cmp bx,01
                      jne read
                      mov cx, 0013H      ;delay for DRQ to go active before
                      again1: loop again1;writing out last byte, 20MHz
                      mov dx, TC_PORT
                      mov bx, last_address
                      mov es, v_es
                      mov al, byte ptr es:[bx]
                      out dx,al
                      jmp done
                read:   mov cx, 0013H       ;delay for DRQ to go active before
                ag2:    loop ag2            ;reading last byte, 20MHz
                      mov dx, TC_PORT
                      in al,dx
                      mov bx,last_address
                      mov es,v_es
                      mov byte ptr es:[bx],al
                      sti
                      done:   pop bx
                      pop dx
                      pop ax
                     }
                     {
                      if(get_fdc_status(myblock) == ok)
                       {
                        orb40(DISK_ID+myblock->fd_drive,media_established);/* break;*/  //
get out with good operation
                       }
                     }
                   }
                  }
                 }
                }
              }
            }
           }
          }
        while (retry(myblock) == error);
```

272339-27

```
   }
    while ((--myblock->fd_try_count > 0) &&
    ((peekb40(DISK_ID+myblock->fd_drive) & media_established) == 0) &&
    ((DISK_STATUS & TIME_OUT) != TIME_OUT));
    ax = (DISK_STATUS << 8) | (calc_sectors(myblock));
    if ((ax & 0xff00) != 0) flags |= 1;
    break;

 /*****************************************
  *   Format Track                        *
  *****************************************/

case FORMAT_TRACK : /* format track */
  myblock->fd_fdc_command = FDC_FORMAT;
  myblock->fd_dma_command = DMA_TX_MODE;
  // if media not established, then set media type = drive type
  if ((peekb40(DISK_ID+myblock->fd_drive) & media_established) == 0)
  pokeb40(DISK_ID+myblock->fd_drive,
    (peekb40(DISK_ID+myblock->fd_drive) & 0x07) |
    ((peekb40(DISK_ID+myblock->fd_drive) & 0x07) << 4));
  MOTOR_STATUS |= 0x80;      /* indicate write operation */
  if (med_change(myblock) == ok)     /* check for media change */
   {
  send_fdc(3);  /* send specify command */
  send_fdc(GetParm(myblock,0));
  send_fdc(GetParm(myblock,1));
  send_rate(myblock);
  if (dma_setup(myblock,es,bx) == ok)
   {
    if (fdc_init(myblock) == ok)
     {
      if (send_fdc(GetParm(myblock,3)) == ok)
       {
        if (send_fdc(GetParm(myblock,4)) == ok)
         {
        if (send_fdc(GetParm(myblock,7)) == ok)
         {
          send_fdc(GetParm(myblock,8)); /* send command */
          get_fdc_status(myblock);
         }
         }
       }
     }
   }
   }
  ax = DISK_STATUS << 8;
  if ((ax & 0xff00) != 0) flags |= 1;
break;

/*****************************************
 *   Read Drive Parameters               *
 *****************************************/

case DISK_PARMS : /* read drive parameters */
  ax = bx = cx = dx = es = 0;    /* reset all registers */
  if (myblock->fd_drive > 0x80)
   {
  ax = (ax & 0xff) | (BAD_CMD << 8); flags |= 1;  /* set return error flag */
   }
  else
   {
  if ((EQUIP_FLAG & 0x01) != 0) /* there are drives present */
   {
    di = dx; // save old dx temporarily
```

272339-28

```
        dx = EQUIP_FLAG >> 6;
        /* return # of drives in dx */
        if (di > dx) /* called for a non-existent drive */
          di = 0; /* return with null parameters */
        else
         {
          di = 0;
          if (peekb40(DISK_ID + myblock->fd_drive) != 0)  /* get parms for this type */
           {
            myblock->fd_drive_type = myblock->fd_media_type =(peekb40(DISK_ID + myblock-
>fd_drive)) & 0x07;
            di = &fdisk_table[myblock->fd_drive_type][myblock->fd_media_type][0];
            myblock->fd_sector_track = peekbcs(&fdisk_table[myblock->fd_drive_type][myblock-
>fd_media_type][DT_SEC_TRK]);
            myblock->fd_max_track    = peekbcs(&fdisk_table[myblock->fd_drive_type][myblock-
>fd_media_type][DT_MAX_TRK]);
            cx = ((myblock->fd_max_track << 8) & 0xff00) | (myblock->fd_sector_track &
0x3f);
            dx = (1 << 8) | dx;
            bx = myblock->fd_drive_type;
            es = bios_cs();       // needs to be our code segment
           }
         }
        }
        }
    break;

    /*****************************************
     *   Read Disk Type                      *
     *****************************************/

    case DISK_TYPE :  /* read disk type */
      ax &= 0x00ff;  /* no drive */
      if ((peekb40(DISK_ID+myblock->fd_drive) & drive_field) != drive_none)
       {
      if ((peekb40(DISK_ID+myblock->fd_drive) & drive_field) == drive_360)
       {
        ax |= 0x0100; /* 40 track, no change line */
       }
      else
       {
        ax |= 0x0200; /* 80 track, change line */
       }
       }
    break;

    /*****************************************
     *   Check for Disk Change               *
     *****************************************/

    case DISK_CHANGE :     /* return change line condition */
      /* if no drive - then return a timeout condition */
      if ((peekb40(DISK_ID+myblock->fd_drive) & drive_field) == drive_none)
       {
      DISK_STATUS |= TIME_OUT;
       }
      else
       {
      /* if 360 k drive - always return disk change */
      if ((peekb40(DISK_ID+myblock->fd_drive) & drive_field) == drive_360)
        DISK_STATUS = MEDIA_CHANGE;
      else
        /* if 720, 1.4, or 2.8 drive - read the change line */
        if (read_dskchng(myblock) != 0 )
```

272339-29

```
          DISK_STATUS = MEDIA_CHANGE;
        }
      ax = (DISK_STATUS << 8) | (ax & 0xff);
      if ((ax & 0xff00) != 0)
      flags |= 1;
    break;

    /*****************************************
     *   Setup Disk Format Type              *
     *****************************************/

    case FORMAT_SET : /* set disk type */
      /* set drive to requested  format/media */
      myblock->fd_drive_type = peekb40(DISK_ID+myblock->fd_drive) & 0x07;
      switch (ax & 0x00ff) /* use requested type for switch */
       {
      case 1: /* 360/360 */
        if(myblock->fd_drive_type == drive_360)
          pokeb40(DISK_ID+myblock->fd_drive,(media_360 | drive_360 | media_established));
      break;

      case 2: /* 360/1.2 */
        if(myblock->fd_drive_type == drive_12)
          pokeb40(DISK_ID+myblock->fd_drive,(media_360 | drive_12 | media_established));
      break;

      case 3: /* 1.2/1.2 */
        if(myblock->fd_drive_type == drive_12)
          pokeb40(DISK_ID+myblock->fd_drive,(media_12 | drive_12 | media_established));
      break;

      case 4: /* 720/720 */
        if(myblock->fd_drive_type == drive_720)
          pokeb40(DISK_ID+myblock->fd_drive,(media_720 | drive_720 | media_established));
      break;

      default:
        DISK_STATUS = BAD_CMD;  /* bad command */
      break;
       }
      ax = (DISK_STATUS << 8) | (ax & 0xff);
      if ((ax & 0xff00) != 0) flags |= 1;
    break;

    /*****************************************
     *   Set Media Type                      *
     *****************************************/

    case SET_MEDIA :      /* set media type */
      // get the max tracks called for
      myblock->fd_max_track = (cx >> 8) | ((cx << 2) & 0x300);
      // get the max sectors called for
      myblock->fd_max_sectors = cx & 0x003f;
      if ((peekb40(DISK_ID + myblock->fd_drive) != 0))
       {
      myblock->fd_drive_type =  (peekb40(DISK_ID + myblock->fd_drive)) & 0x07;
      // use the transition table for this drive
      for (myblock->fd_i= 0;peekbcs(&transition_table[myblock->fd_drive_type][myblock-
>fd_i]) != 0;myblock->fd_i++)
       {
        myblock->fd_media_type = peekbcs(&transition_table[myblock->fd_drive_type][myblock-
>fd_i]);
        if ((myblock->fd_max_sectors == peekbcs(&fdisk_table[myblock-
>fd_drive_type][myblock->fd_media_type][DT_SEC_TRK])) && \
```

272339-30

```
          (myblock->fd_max_track == peekbcs(&fdisk_table[myblock->fd_drive_type][myblock-
>fd_media_type][DT_MAX_TRK])))
         {
          // return the disk parms ptr to the caller
          di = &fdisk_table[myblock->fd_drive_type][myblock->fd_media_type][0];
          es = bios_cs();
          // set the disk_id to the established supported media
          pokeb40(DISK_ID+myblock->fd_drive,myblock->fd_drive_type | (myblock->fd_media_type
<< 4) |
          drive_established | media_established);
          // need a break to say OK here !!!
          goto set_media_exit;
         }
       }
      if (peekbcs(&transition_table[myblock->fd_drive_type][myblock->fd_i]) == 0)
      DISK_STATUS = MED_NOT_FND;
       }
      set_media_exit:
      ax = (DISK_STATUS << 8) | (ax & 0xff);
      if ((ax & 0xff00) != 0)
      flags |= 1;
    break;

    /*****************************************
     *   Default                             *
     *****************************************/

    default : /* invalid function code - return bad code */
      DISK_STATUS = BAD_CMD;
      ax = (BAD_CMD << 8) | (ax & 0xff);
      flags |= 1;  /* set return error flag */
    }  /* end of switch */
  }
 MOTOR_COUNT = GetParm(myblock,2); /* set motor count */
 outpw(DOR_PORT,MOTOR_OFF);
 release_block(myblock);
}  /* end of disk_io */

/*************************************************************************
 *                                                                       *
 *     Reset the disk system                                             *
 *                                                                       *
 *************************************************************************/

void FDC_reset(fdisk_regs *myblock)
{
 unsigned i;
 unsigned char status;

 _disable();
 status = (MOTOR_STATUS << 4) | 8;
 // if a motor is on, set the drive select bits accordingly
 switch (status & 0xf0)
  {
   /* motor 2 on */
   case 0x20: status |= 0x01; break;
   /* motor 3 on */
   case 0x40: status |= 0x02; break;
   /* motor 4 on */
   case 0x80: status |= 0x03; break;
  }
 outp(DOR_PORT,status);  /* reset disk controller */
 SEEK_STATUS = 0;
 outp(DOR_PORT,status | 0x04); // reset the reset bit
```

272339-31

```
 _enable();    /* enable interrupts */
 /* wait for interrupt */
 if (wait_int() == error) { DISK_STATUS |= BAD_FDC; return; }
   for (i = 0; i < 4; i++)   /* number of drives */
    {
     /* send command */
  if (send_fdc(8) == error) { DISK_STATUS |= BAD_FDC; return; }
  /* check results */
  if (results() != ok) { DISK_STATUS |= BAD_FDC; return; }
  if (peekb40(FDC_STATUS) != (i | 0xc0)) { DISK_STATUS |= BAD_FDC; return; }
 }
 send_specify_command(myblock);
}

/*************************************************************************
 *                                                                       *
 *     When the hardware interrupt occurs from the floppy disk,          *
 *     this function sets bit 7 in the seek_status flag and sends        *
 *     an end-of-interrupt command to the 8259.                          *
 *                                                                       *
 *************************************************************************/

void interrupt far fdisk_isr (void)
{
 SEEK_STATUS |= INT_FLAG;
 outpw(EOI,INT0_EOI);          /* INT0 eoi */
}

/*************************************************************************
 *                                                                       *
 *       Send a byte to the FDC.                                         *
 *                                                                       *
 *************************************************************************/

unsigned send_fdc(unsigned char parm)
{
 unsigned long i = TIMER_LONG+20;       /* Wait one second */
 unsigned j,z;

 do
  {
   if ((inp(MSR_PORT) & 0xc0) == RQM)
    {
     outp(DATA_PORT,parm);
     delay_call(0,1); /* delay at least 50us */
     return (ok);
    }
  }
 while (TIMER_LONG<i);
 DISK_STATUS |= TIME_OUT;
 return (error);
}

/*************************************************************************
 *                                                                       *
 *          Get the indexed value from the disk parameter table.         *
 *                                                                       *
 *************************************************************************/

unsigned GetParm(fdisk_regs *myblock,unsigned char index)
{
 return(peekbcs(&fdisk_table[myblock->fd_drive_type][myblock->fd_media_type][index]));
}
```

272339-32

```
/*************************************************************************
 *                                                                       *
 *                  Send the specify command to the FDC.                 *
 *                                                                       *
 *************************************************************************/

void send_specify_command (fdisk_regs *myblock)
{
 send_fdc(3);
 send_fdc(GetParm(myblock,0));
 send_fdc(GetParm(myblock,1));  /* send command */
}

/*************************************************************************
 *                                                                       *
 *      Turn motor on and wait for motor start up time if this is a      *
 *      write operation. Proceed immediately if a read operation.        *
 *      1. save the last state of the motor.                             *
 *      2. turn on the selected motor.                                   *
 *      3. if not a write, exit immediately.                             *
 *      3. if a write, if the same motor was already on, exit, else      *
 *         wait for the delay.                                           *
 *                                                                       *
 *************************************************************************/

void motor_on(fdisk_regs *myblock)
{
 unsigned char this_motor, last_motor, wait;
 unsigned i,j,y,z;
 _disable();
 MOTOR_COUNT = 0xff;
 /* hit timer with max on-count */
 last_motor = MOTOR_STATUS & 0x0f;
 wait = MOTOR_STATUS & 0x80;
 MOTOR_STATUS = this_motor = (1 << myblock->fd_drive);
 outp(DOR_PORT,(this_motor << 4) | 0x0c | myblock->fd_drive);
 _enable();
 if((last_motor != this_motor) && (wait != 0))
  {
   delay_call(1,GetParm(myblock,10));
   MOTOR_COUNT = 0xff;
  }
}

/*************************************************************************
 *                                                                       *
 *      Wait for the hardware interrupt to occur. Time-out and return    *
 *      if no interrupt.                                                 *
 *                                                                       *
 *************************************************************************/

unsigned wait_int (void)
{

 unsigned long i=TIMER_LONG+20;       /* this time out should be 3 secs */
 _disable();
 outpw(IMASK,inpw(IMASK) & ~BIT4);    /* temp (fix to unmask) mask interrupt */
 _enable();
 do
  {
   if((SEEK_STATUS & INT_FLAG) != 0) SEEK_STATUS &= ~INT_FLAG; return (ok);
  }
 while (TIMER_LONG<i);
 DISK_STATUS |= TIME_OUT; SEEK_STATUS &= ~INT_FLAG;
```

272339-33

```
 return (error);
}

/*************************************************************************
 *                                                                       *
 *          Send the data-transfer-rate command to controller            *
 *                                                                       *
 *************************************************************************/

void send_rate(fdisk_regs *myblock)
{
 outp(DRR_PORT,peekbcs(&fdisk_table[peekb40(DISK_ID+myblock->fd_drive) & 0x07]
 [(peekb40(DISK_ID+myblock->fd_drive) >> 4) & 0x07][DT_RATE]));
}

/*************************************************************************
 *                                                                       *
 *   Process the interrupt received after recalibrate or seek.          *
 *                                                                       *
 *************************************************************************/

unsigned chk_stat_2 (void)
{
 if (wait_int() == error) return (error);
 if (send_fdc(8) == error) return (error);
 if (results() == error) return (error);
 if ((peekb40(FDC_STATUS) & 0x60) == 0x60)  /* normal termination */
 {
  DISK_STATUS |= BAD_SEEK;
  return (error);
 }
 return (ok);
}

/*************************************************************************
 *                                                                       *
 *     Initialize dma controller for read, write, verify operations.     *
 *                                                                       *
 *************************************************************************/

unsigned dma_setup(fdisk_regs *myblock,unsigned es, unsigned bx)
{
 _disable();    /*disable interrupts */
 /* calculate upper 4 nibbles of buffer address */
 myblock->dma_address = es + (bx >>4);
 /* store upper nibble of address */
 myblock->nibble =myblock->dma_address >> 12;
 /* calculate and store lower 4 nibbles of address */
 myblock->word = (((myblock->dma_address << 4) & 0xfff0) | (bx & 0x000f));
 if  (myblock->fd_dma_command == DMA_TX_MODE)     /*dma write to disk */
  {
   /* output address of data buffer to the DMA source pointer registers */
   outp(D0SRCH,myblock->nibble);
   outp(D0SRCL,myblock->word);
   /* output address of FDC data port to the DMA destination pointers */
   outp(D0DSTH,0);
   outp(D0DSTL,DACK_PORT);
   /* form control register word:
      destination synchronization, incremented memory source, non-incremented I/O
      destination, terminal count and interrupt enabled, byte transfers */
   myblock->dma_control = 0x1686;
  }
 else      /* read from disk to memory */
  {
```

272339-34

```
    /* output address of data buffer to the DMA destination pointer registers */
    outp(D0DSTH,myblock->nibble);
    outp(D0DSTL,myblock->word);
    /* output address of FDC data port to the DMA source pointers */
    outp(D0SRCH,0);
    outp(D0SRCL,DACK_PORT);
    /* form control register word:
    source synchronization, non- incremented I/O source, incremented memory destination,
terminal count and interrupt enabled, byte transfers */
    myblock->dma_control = 0xA246;
   }
  /* calculate dma transfer length */
  myblock->length = ((myblock->fd_nr_sectors * 128) << GetParm(myblock,3))-1; /*-1 taken
out*/
  outp(D0TC,myblock->length);     /* load transfer count register */
  /* calculate value of the last byte to be transferred */
  myblock->seg_check= 0;
  myblock->seg_check = ((unsigned long) bx+ (unsigned long )myblock->length);
  last_address = myblock->seg_check;
  /* if last byte to be transferred is in a different segment than the first, update the
segment variable */
  if (myblock->seg_check & 0xffff0000)
    ++v_es;
  /* load the value at last_address into the variable last_byte */
  __asm
  {
   push es
   push ax
   push bx
   mov es, v_es
   mov bx, last_address
   pop bx
   pop ax
   pop es
  }
  _enable();
  outp(D0CON,myblock->dma_control);  /* load DMA control register */
  /* DMA channel is now armed, a request on DRQ2 will now result in transfers */
  if (myblock->seg_check  & 0xffff0000)
   {
    DISK_STATUS = DMA_BOUNDARY;
    return (error);
   }
  return (ok);
}

/*************************************************************************
 *                                                                       *
 *       Move the head to the selected track.                            *
 *                                                                       *
 *************************************************************************/

unsigned seek(fdisk_regs *myblock)
 {
  if ((SEEK_STATUS & (1 << myblock->fd_drive)) == 0) /* need recal */
   {
    SEEK_STATUS |= (1 << myblock->fd_drive); /* mark recal will be done */
    /* try 2 attemps at recalibrate, then if failed, return error */
    if (recal(myblock) == error)
     {
      DISK_STATUS = 0;
      if (recal(myblock) == error) return (error);
     }
    pokeb40(LAST_TRACK+(myblock->fd_drive),0); /* clear track number */
```

272339-35

```
    /* if we want track zero, then just wait for head and exit */
    if (myblock->fd_track == 0) { wait_for_head(myblock); return (ok); }
   }
  if (peekbcs(&fdisk_table[myblock->fd_drive_type][myblock->fd_media_type][DT_STEP]) != 0)
    myblock->fd_track *= 2;
  if (peekb40(LAST_TRACK+myblock->fd_drive) == myblock->fd_track)
    return(ok);
  /* update new position */
  pokeb40(LAST_TRACK+myblock->fd_drive,myblock->fd_track);
  if (send_fdc(0x0f) == error) return (error);
  if (send_fdc(myblock->fd_drive) == error) return (error);
  if (send_fdc(myblock->fd_track) == error) return (error);
  if (chk_stat_2() == error) return (error);
  wait_for_head(myblock);
  return (ok);
 }

/************************************************************************
 *                                                                      *
 *         Recalibrate the drive.                                       *
 *                                                                      *
 ************************************************************************/

unsigned recal(fdisk_regs *myblock)
 {
  if (send_fdc(7) == error) return (error);
  if (send_fdc(myblock->fd_drive) == error) return (error);
  /* send command */
  if (chk_stat_2() == error) return (error);
  /* return with error */
  return (ok);       /* return with no error */
 }

/************************************************************************
 *                                                                      *
 *              Determine whether a retry is necessary.                 *
 *              Returning an OK says don't retry any more               *
 *              Returning an ERROR says retry again                     *
 *                                                                      *
 ************************************************************************/

unsigned retry(fdisk_regs *myblock)
 {
  unsigned char media_type;
  /* if operation timed out - say no retry */
  if ((DISK_STATUS & TIME_OUT ) == TIME_OUT) return (ok);
  /* if media is established - say no retry */
  if ((peekb40(DISK_ID+myblock->fd_drive) & media_established) != 0) return (ok);
  /* we have to step through the media */
  // get the next possible media type for this drive type
  media_type = peekbcs(&transition_table[myblock->fd_drive_type][++myblock-
>fd_media_index]);
  if (media_type == media_none)  // then at end of possibles
   {
    // dis-establish media
    // return - no more
    andb40(DISK_ID+myblock->fd_drive,~(media_established | media_field));
    return(ok);
   }
  // insert the new media type into the DISK_ID and return saying try again
  pokeb40(DISK_ID+myblock->fd_drive,(peekb40(DISK_ID+myblock->fd_drive) & ~media_field) |
  (media_type << 4));
  DISK_STATUS = 0;
  pokeb40(FDC_STATUS,0);
```

272339-36

```
  return(error); /* return saying retry */
 }

/*************************************************************************
 *                                                                       *
 *      Read anything from the controller following an interrupt.        *
 *      This may include up to seven bytes of status.                    *
 *                                                                       *
 *************************************************************************/

unsigned results(void)
{
 unsigned count = 0;
 unsigned long time;
 unsigned long j,z;
 loop_label1:
 /* time = TIMER_LONG;*/
 time = TIMER_LONG+2;
 loop_label2:
 if ((inp(MSR_PORT) & 0xc0) != 0xc0) /* data not ready from FDC */
  {
   if(TIMER_LONG < time) goto loop_label2;
     DISK_STATUS |= TIME_OUT;
     return (error);
  }
 pokeb40(FDC_STATUS+count,inp(DATA_PORT)); /* save status */
 count++;
 if ((inp(MSR_PORT) & BUSY) == 0) return (ok);
 if(count < 7) goto loop_label1;
 DISK_STATUS |= BAD_FDC;
 return(error);
}

/*************************************************************************
 *                                                                       *
 *     Purge the FDC of any status it is waiting to send.                *
 *                                                                       *
 *************************************************************************/

void purge_fdc (void)
 {
  unsigned count  = 0;
  unsigned j,z;
  while (((inp(MSR_PORT) & 0xc0) == 0xc0) && (count++ < 25))
   {
    inp(DATA_PORT);  /* read status */
    delay_call(0,1);   /* Delay at least 50us */
   }
 }


/*************************************************************************
 *                                                                       *
 *       Read the state of the disk change line.                         *
 *                                                                       *
 *************************************************************************/

unsigned char read_dskchng(fdisk_regs *myblock)
 {
  motor_on(myblock);   /* turn motor on */
  return (inp(DIR_PORT) & DSKCHANGE_BIT);
 }

/*************************************************************************
```

272339-37

```
 *                                                                          *
 *    Execute the FDC read id command.                                      *
 *                                                                          *
 ****************************************************************************/

unsigned read_id (fdisk_regs *myblock)
 {
  if (send_fdc(FDC_READID) == error) return(error);
  if (send_fdc(myblock->fd_head << 2 | myblock->fd_drive) == error) return (error);
  return (get_fdc_status(myblock));
 }

/****************************************************************************
 *                                                                          *
 *       Wait for the head to settle after a seek.        *                 *
 *                                                                          *
 ****************************************************************************/

void wait_for_head(fdisk_regs *myblock)
 {
  unsigned char wait;
  unsigned i,j;

  wait = GetParm(myblock,9);   /* get head settle */
  if ((MOTOR_STATUS & 0x80) != 0)  // if write
   {     /* yes */
    if (wait == 0)   /* wait zero ? */
     {     /* yes */
      if ((peekb40(DISK_ID+(myblock->fd_drive)) >> 4) & 0x07)
      wait = 0x14;   /* default 360 head time */
      else
      wait = 0x0f;   /* default others head time */
     }
   }
  else
   {
    if (wait == 0)   /* is wait zero ? */
      return;    /* yes, return */
   }
  delay_call(1,wait); /* Delay n milliseconds */
 }

/****************************************************************************
 *                                                                          *
 *      Checks for a media change, reset media changes and check            *
 *      media changes.                                                      *
 *      Returns:                                                            *
 *      ok = media not changed, error = media changed                       *
 *                                                                          *
 ****************************************************************************/

unsigned med_change(fdisk_regs *myblock)
 {
  unsigned TrackSave = myblock->fd_drive;
  if (read_dskchng(myblock) == 0) return (ok);
  /* clear media established and media type */
  andb40(DISK_ID+myblock->fd_drive,~media_established);
  _disable();
  MOTOR_STATUS &= ~(1 << myblock->fd_drive);
  /* turn off motor status */
  _enable();
  motor_on(myblock);
  FDC_reset(myblock);
  myblock->fd_track = 0;
```

272339-38

```
  seek(myblock);
  myblock->fd_track = 1;
  seek(myblock);
  DISK_STATUS = MEDIA_CHANGE;
  if (read_dskchng(myblock) != 0) DISK_STATUS = TIME_OUT;
  myblock->fd_drive = TrackSave;
  return (error);
 }


/************************************************************************
 *                                                                      *
 *      Seek to the requested track and initialize the controller       *
 *                                                                      *
 ************************************************************************/

unsigned fdc_init(fdisk_regs *myblock)
 {
  motor_on(myblock);
  if (seek(myblock) == error) return (error);
  if (send_fdc(myblock->fd_fdc_command) == error) return(error);
  if (send_fdc(((myblock->fd_head <<2) & BIT2) | myblock->fd_drive) == error)
  return(error);
  return (ok);
 }

/************************************************************************
 *                                                                      *
 *   Wait until an operation is complete, then accept the status        *
 *   from the controller.                                               *
 *                                                                      *
 ************************************************************************/

unsigned get_fdc_status(fdisk_regs *myblock)
 {
  myblock->time_out_flag = wait_int();
  if (results() == error) return (error);
  if (myblock->time_out_flag == error)
   {
    if (DISK_STATUS == 0) return(ok); else return(error);
   }
  if ((peekb40(FDC_STATUS) & 0xc0) == 0)
   {
    if (DISK_STATUS == 0) return(ok); else return(error);
   }
  if ((peekb40(FDC_STATUS) & 0xc0) != 0x40) { DISK_STATUS |= BAD_FDC; return(error); }
  myblock->error_byte = peekb40(FDC_STATUS + 1);
  /* get controller status */
  if ((myblock->error_byte & BIT7) != 0) {DISK_STATUS |= RECORD_NOT_FND;}
  else if ((myblock->error_byte & BIT5) != 0) {DISK_STATUS |= BAD_CRC;}
  else if ((myblock->error_byte & BIT4) != 0) {DISK_STATUS |= BAD_DMA;}
  else if ((myblock->error_byte & BIT2) != 0) {DISK_STATUS |= RECORD_NOT_FND;}
  else if ((myblock->error_byte & BIT1) != 0) {DISK_STATUS |= WRITE_PROTECT;}
  else if ((myblock->error_byte & BIT0) != 0) {DISK_STATUS |= BAD_ADDR_MARK;}
  else   {DISK_STATUS |= BAD_FDC;}
  if (DISK_STATUS != 0) return(error);
  return(ok);
 }


/************************************************************************
 *                                                                      *
 *      calculate number of sectors that were actually transferred.     *
 *      returns:  number of sectors transferred                         *
```

272339-39

```
 *                                                                          *
 ****************************************************************************/

#define end_head peekb40(FDC_STATUS + 4)
#define end_track peekb40(FDC_STATUS + 3)
#define sectors_track GetParm(myblock,4)

unsigned char calc_sectors (fdisk_regs * myblock)
 {
  unsigned end_sector = peekb40(FDC_STATUS + 5);

  if (DISK_STATUS != 0) return (0);
  if (end_head != myblock->fd_head)
      { end_sector += sectors_track; }
  else
   {
    if (end_track != myblock->fd_track)
     {
     end_sector += (sectors_track + sectors_track); }
     }
  return(end_sector - myblock->fd_sector);
 }

/****************************************************************************
 *                                                                          *
 *                 Delay Routine                                            *
 *                                                                          *
 ****************************************************************************/

void delay_call(unsigned res,unsigned count)
 {
  DELAY_LONG=0;
  if (res==0)                       /* 50us delay*/
    while (DELAY_LONG<count)
      {}
  else
   {
    count=(count*20);               /*1 millisec delay*/
    while (DELAY_LONG<count*20)
      {}
   }
  return;
 }

/****************************************************************************/
```

272339-40

```
;*************************************************************************
;*                                                                       *
;* Copyright (c) FOSCO 1988, 1989, 1991 - All Rights Reserved            *
;*                                                                       *
;* Module Name: 80C186 Floppy Disk Driver Assembly Language functions    *
;*                                                                       *
;* Version: 1.04b                                                        *
;*                                                                       *
;* Author: FOSCO                                                         *
;*                                                                       *
;* Modifications done by: BRENDAN RUIZ(Intel), ERIC AUZAS(Intel)         *
;*                                                                       *
;* Date: 10-19-92                                                        *
;*                                                                       *
;* Filename: ATTOP.ASM                                                   *
;*                                                                       *
;* Language: MS MASM 6.0                                                 *
;*                                                                       *
;* Functional Description:                                               *
;*                                                                       *
;*       This module serves the purpose of holding assembly language     *
;*       functions.                                                      *
;*                                                                       *
;* 1.04b 10-19-92                                                        *
;*  deleted all un-necessary functions not needed for the Floppy Disk    *
;*  Controller driver.                                                   *
;*                                                                       *
;*************************************************************************

      .186
      .model small,C
      .xlist

_scratch segment word public 'scratch'
_scratch ends
_text   segment word public 'code'
_text   ends
_data   segment word public 'data'
_data   ends
_const   segment word public 'const'
_const   ends
_bss    segment word public 'bss'
_bss    ends
_fill   segment word public 'fill'
_fill   ends
_stack  segment word stack 'stack'
_stack  ends
_atext  segment word public 'acode'
_atext  ends


dgroup group _scratch,_STACK,_data,_const,_bss,_fill
      assume cs:_text, ds:dgroup, ss:dgroup

_scratch segment word public 'scratch'

public  start_block
public  current_open
public  block_beg
public  block_end
public  end_block

start_block     dw  ?
current_open    dw  ?
```

272339-41

```
end_block       dw  ?

block_beg       label   word
       dw   1024 dup(?)
block_end       label   word

_scratch ends

;--- clear block for release_block in atmisc ------------

_text  segment word public 'code'
      assume cs:_text

;*********************************************************************
;* void clear_block(pointer, count);                                 *
;*********************************************************************

clear_block proc uses ax es di cx,block_ptr:word,count:word
      pushf
      xor     ax,ax
      mov     es,ax
      mov     ax,es:[0400eh]   ; get sys seg ptr
      mov     es,ax
      mov     di,block_ptr
      mov     cx,count
      xor     ax,ax
      cld
      cli
      rep     stosb
      popf
      ret
clear_block     endp


seg_40_constant dw 400h

;*********************************************************************
;* void setds_system_segment(void);                                  *
;*********************************************************************

setds_system_segment proc
      mov     ds,cs:seg_40_constant
      mov     ds,ds:[0eh]
      ret
setds_system_segment endp

;*********************************************************************
;* void sys_int(char number,reg_block);                              *
;*                                                                   *
;* This version assumes the regs are in the system segment.          *
;* Note that is also requires a slightly different regs structure    *
;* than int86 or int86x because it always passes es and ds.          *
;*********************************************************************

axoff   equ     4
cxoff   equ     6
dxoff   equ     8
sioff   equ     10
dioff   equ     12
bpoff   equ     14
bxoff   equ     16
dsoff   equ     18
esoff   equ     20
floff   equ     22
```

272339-42

```
jmp_off equ     24
jmp_seg equ     26
code_string_location    equ     28


code_string     proc far
      lds     bx,cs:[bx+bxoff]
      int     0
number_offset = $-code_string-1
      ret
code_string_length = $-code_string
code_string     endp

;*************************************************************************
;* void sys_int(number, regs);                                           *
;*************************************************************************

sys_int proc uses si di ax bx cx dx es ds,      number:word,regs_block:word

      mov     bx,regs_block

      ; move the code string image
      ; get count of bytes to move
      mov     cx,code_string_length
      ; point to org of code_string
      mov     si,offset code_string
      ; build ptr to destination
      mov     di,bx
      add     di,offset code_string_location
      ; set segment for destination
      push    ds
      pop     es
      ; now do the copy
      rep     movs    byte ptr es:[di], byte ptr cs:[si]

      ; load the correct interrupt number
      mov     ax,number
      mov     [bx+code_string_location+number_offset],al

      ; load the jump vector
      mov     word ptr [bx+jmp_off],code_string_location
      add     [bx+jmp_off],bx
      mov     [bx+jmp_seg],ds

      ; load the registers
      mov     ax,[bx+axoff]
      mov     cx,[bx+cxoff]
      mov     dx,[bx+dxoff]
      mov     si,[bx+sioff]
      mov     di,[bx+dioff]
      mov     es,[bx+esoff]

      ; save the real bp for the implied LEAVE instruction
      push    bp
      mov     bp,[bx+bpoff]
      push    ds
      push    bx
      call    dword ptr ds:[bx+jmp_off]

      ; now we have to swap out ds:bx with the stack-saved ones

      push    bx              ; save returned ds:bx
      push    ds
      push    bp              ; save bp - will be used as ptr
```

272339-43

```
        mov     bp,sp
        lds     bx,[bp+6]       ; re-load myblock ds:bx

        pop     ds:[bx+bpoff]   ; restore returned bp
        pop     ds:[bx+dsoff]
        pop     ds:[bx+bxoff]

        pop     bp              ; discard old ds:bx slots
        pop     bp              ; without affecting flags

              ; bp is now pre-interrupt value
        pop     bp

              ; store the registers

        mov     [bx+esoff],es
        mov     [bx+axoff],ax
        mov     [bx+cxoff],cx
        mov     [bx+dxoff],dx
        mov     [bx+sioff],si
        mov     [bx+dioff],di
        pushf   ; save flags
        pop     [bx+floff]

        ret
sys_int endp

;************************************************************************
;*          void get_seg(unsigned offset)                               *
;************************************************************************

get_seg proc arg1:dword
        mov ax,word ptr arg1[2]
        ret
get_seg endp

;************************************************************************
;*          void move_system_segment(unsigned offset)                                  *
;************************************************************************

move_system_segment proc uses ax dx ds
        cli
        mov     ax,0400h
        mov     dx,ds
        mov     ds,ax
        mov     ds:[0eh],dx
        sti
        ret
move_system_segment endp

;************************************************************************
;*       unsigned bios_cs(void);                                        *
;*                                                                      *
;* return the bios code segment value in AX.                            *
;* This function is coded this way so the Bios may be operated at       *
;* segments other than F000.                                            *
;************************************************************************

bios_cs proc
        mov     ax,cs
        ret
bios_cs endp

;-------- these are non-intrinsic in/out functions ---
```

272339-44

```
;*********************************************************************
;* void inp(unsigned port, char value);                              *
;*********************************************************************

inp     proc    uses dx,port:word
      mov     dx,port
      in      al,dx
      xor     ah,ah
      ret
inp     endp

;*********************************************************************
;* void inpw(port, value);                                           *
;*********************************************************************

inpw    proc    uses dx,port:word
      mov     dx,port
      in      ax,dx
      ret
inpw    endp

;*********************************************************************
;* void outp(unsigned port, char value);                             *
;*********************************************************************

outp    proc    uses ax dx,port:word,value:word
      mov     dx,port
      mov     ax,value
      out     dx,al
      ret
outp    endp

;*********************************************************************
;* void outpw(port, value);                                          *
;*********************************************************************


outpw    proc    uses ax dx,port:word,value:word
      mov     dx,port
      mov     ax,value
      out     dx,ax
      ret
outpw    endp

;*********************************************************************
;*              void _enable(void);                                  *
;*********************************************************************

_enable proc
      sti
      ret
_enable endp

;*********************************************************************
;*              void _disable(void);                                 *
;*********************************************************************

_disable        proc
      cli
      ret
_disable        endp
```

272339-45

```
;--- These are additional peek and poke functions ---

;*************************************************************************
;* char peekb40(unsigned offset);                                        *
;*************************************************************************

peekb40 proc uses bx es,offptr:word
      mov     bx,400h
      mov     es,bx
      mov     bx,offptr
      mov     al,es:[bx]
      xor     ah,ah
      ret
peekb40 endp

;*************************************************************************
;* unsigned peekbcs(unsigned offset);                                    *
;*************************************************************************

peekbcs proc uses bx,offptr:word
      mov     bx,offptr
      mov     al,ds:[bx]
      xor     ah,ah
      ret
peekbcs endp

;*************************************************************************
;* void pokeb40(unsigned offset, char value);                            *
;*************************************************************************

pokeb40 proc uses ax bx es,offptr:word,value:word
      mov     bx,400h
      mov     es,bx
      mov     bx,offptr
      mov     ax,value
      mov     es:[bx],al
      ret
pokeb40 endp

;*************************************************************************
;* void andb40(unsigned offset, char value);                             *
;*************************************************************************

andb40 proc uses ax bx es,offptr:word,value:word
      mov     bx,400h
      mov     es,bx
      mov     bx,offptr
      mov     ax,value
      and     es:[bx],al
      ret
andb40 endp

;*************************************************************************
;* void orb40(unsigned offset, char value);                              *
;*************************************************************************

orb40   proc uses ax bx es,offptr:word,value:word
      mov     bx,400h
      mov     es,bx
      mov     bx,offptr
      mov     ax,value
      or      es:[bx],al
      ret
orb40   endp
```

272339-46

```
;--- These are the standard peeks, pokes ----

;*********************************************************************
;* unsigned peek(unsigned segment, unsigned offset)                  *
;*********************************************************************

peek    proc uses bx es,segptr:word,offptr:word
      mov     es,segptr
      mov     bx,offptr
      mov     ax,es:[bx]
      ret
peek    endp

;*********************************************************************
;* void poke(segment, offset, value);                                *
;*********************************************************************

poke    proc uses ax bx es,segptr:word,offptr:word,value:word
      mov     es,segptr
      mov     bx,offptr
      mov     ax,value
      mov     es:[bx],ax
      ret
poke    endp

;*********************************************************************
;* char peekb(unsigned segment, unsigned offset);                    *
;*********************************************************************

peekb   proc uses bx es,segptr:word,offptr:word
      mov     es,segptr
      mov     bx,offptr
      mov     al,es:[bx]
      xor     ah,ah
      ret
peekb   endp

;*********************************************************************
;* void pokeb(unsigned segment, unsigned offset, char value);        *
;*********************************************************************

pokeb   proc uses ax bx es,segptr:word,offptr:word,value:word
      mov     es,segptr
      mov     bx,offptr
      mov     ax,value
      mov     es:[bx],al
      ret
pokeb   endp


_text  ends
      end
```

272339-47

```
/*************************************************************************
*                                                                        *
* Copyright (c) FOSCO 1988, 1989 - All Rights Reserved                   *
*                                                                        *
* Module Name: AT Bios h file of standard definitons                     *
*                                                                        *
* Version: 2.00                                                          *
*                                                                        *
* Author: FOSCO                                                          *
*                                                                        *
* Modifications done by: BRENDAN RUIZ(Intel), ERIC AUZAS(Intel)          *
*                                                                        *
* Date: 10-19-92                                                         *
*                                                                        *
* Filename: ATKIT.H                                                      *
*                                                                        *
* Language: Microsoft C 7.0                                              *
*                                                                        *
* Functional Description:                                                *
*                                                                        *
*  This file is used for the standard header for bios 'C' programs.      *
*  It is the only header file that should be #defined in the Bios        *
*  modules. Other headers may contain conflicting definitions.           *
*                                                                        *
*       The Bios does not use any standard library for two main          *
*        reasons:                                                        *
*               1. The linker attempts to place library routines at      *
*                  the top of the segment, which must contain some       *
*                  fixed code.                                           *
*                                                                        *
*               2. The standard libraries assume that the program        *
*                  is running under DOS (which is not the case for       *
*                  Bios), and may make DOS references.                   *
*                                                                        *
*       The file Biostop.asm contains the assembly-language functions    *
*       required for the Bios.                                           *
*                                                                        *
* Version History:                                                       *
* 1.01-1.02                                                              *
*  ATBIOS Kit                                                            *
*                                                                        *
* 2.00                                                                   *
*  Modified for 80C186EA/XL Floppy Disk Controller                       *
*                                                                        *
*************************************************************************/


extern void andb40(unsigned,unsigned char);
void orb40(unsigned,unsigned char);
unsigned get_seg(void);
unsigned bios_cs(void);
void set_vector(unsigned,unsigned,unsigned *);
void lbyte(unsigned char);
void lword(unsigned);
void write_string(unsigned char *);
void bios_unsigned(unsigned,unsigned *,unsigned *);
void move_system_segment(void);
unsigned acquire_scratch_block(unsigned,unsigned);
void release_block(unsigned);

enum { error,ok };

#define sys_seg_size 4096       // size of the system segment
```

272339-48

```
#define BIT15  0x8000
#define BIT14  0x4000
#define BIT13  0x2000
#define BIT12  0x1000
#define BIT11  0x0800
#define BIT10  0x0400
#define BIT9   0x0200
#define BIT8   0x0100
#define BIT7   0x0080
#define BIT6   0x0040
#define BIT5   0x0020
#define BIT4   0x0010
#define BIT3   0x0008
#define BIT2   0x0004
#define BIT1   0x0002
#define BIT0   0x0001
#define FLOPPY BIT7
#define interrupt_registers \
      unsigned es,unsigned ds, unsigned di, unsigned si,unsigned bp, unsigned sp, \
      unsigned bx,unsigned dx, unsigned cx, unsigned ax,unsigned ip, unsigned cs, unsigned
flags

// This definition provides a simplified means to set the interrupt vector to the service
routine.

#define link_interrupt(level,name) set_vector(level,bios_cs(),name)

//-------------------------------------------------------------------------------------------
--------

#define variables typedef struct {\
unsigned length_tag; \
unsigned user_id;\
unsigned ax;\
unsigned cx;\
unsigned dx;\
unsigned si;\
unsigned di;\
unsigned bp;\
unsigned bx;\
unsigned ds;\
unsigned es;\
unsigned flags;\
unsigned jmp_off;\
unsigned jmp_seg;\
unsigned code_string[18]; \

#define end_variables unsigned char size; }

// This definition sizes the block acquire to include the variables specified
// in the variable declaration

#define acquire_block(id) acquire_scratch_block(id, \
(&myblock->size - &myblock->length_tag) + \
(16-((&myblock->size - &myblock->length_tag) % 16)))

//-----------------------------------------------------
// definitions of variables in data segment segment 40h

#define SYSTEM_SEGMENT_PTR (*((unsigned far *) 0x400e))
#define EQUIP_FLAG (*((unsigned far *) 0x4010))
#define SEEK_STATUS (*((unsigned char far *) 0x403e))
#define MOTOR_STATUS (*((unsigned char far *) 0x403f))
#define MOTOR_COUNT (*((unsigned char far *) 0x4040))
```

272339-49

```
#define DISK_STATUS (*((unsigned char far *) 0x4041))
#define TIMER_LONG (*((unsigned long far *) 0x406c))
#define DELAY_LONG (*((unsigned long far *) 0x4070))
#define TIMER_STEP (*((unsigned long far *) 0x4074))

/* 490-493 are floppy disk drive and media type bytes */
/* 494-497 are floppy disk drive current track positions */


/*======================== end of bioskit.h ==================================*/
```

272339-50